VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Aplikace metod z teorie grafů pro analýzu Petriho sítí

Application of the Methods of Graph Theory to Analyze Petri Nets

2017

Petr Šlachta

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petr Šlachta**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Aplikace metod z teorie grafů pro analýzu Petriho sítí
Application of the Methods of Graph Theory to Analyze Petri Nets

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem této práce je rozšířit nástroj, který umožňuje vytvořit, editovat a dále analyzovat P/T Petriho síť. Analýza Petriho sítí bude zaměřena na vlastnosti, které lze detekovat pomocí grafových algoritmů aplikovaných přímo na Petriho síť nebo na graf dosažitelnosti (či pokrytí) Petriho sítě.

Dílčí body práce budou:
1. Seznamte se a vytvořte přehled metod využívajících grafových algoritmů pro analýzu PN.
2. Seznamte se s nástrojem vyvíjeným na katedře pro tvorbu, editaci a analýzu P/T Petriho sítí.
3. Implementujte vybrané grafové metody pro analýzu P/T PN do stávajícího programu.
4. Rozšiřte vizualizační možnosti tohoto nástroje pro výsledky získané grafovými algoritmy.

Seznam doporučené odborné literatury:

[1] Jaroslav Markl: Petriho sítě I.
[2] Reisig, W. (2012). Petri nets: an introduction (Vol. 4). Springer Science & Business Media.
[3] Tantau, T. (2013, January). Graph drawing in TikZ. In Graph Drawing (pp. 517-528). Springer Berlin Heidelberg.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Pavla Dráždilová, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2017

.....

Rád bych na tomto místě poděkoval své vedoucí diplomové práce, paní Mgr. Pavle Dráždilové, Ph.D., za její rady, pomoc a věnovaný čas během tvorby této práce.

Abstrakt

V této práci popisujeme rozšíření nástroje pro editaci a analýzu P/T Petriho sítí, vyvíjeného na katedře informatiky. Nástroj je tvořen webovým grafickým editorem, analytickým serverem a databázovým úložištěm. V úvodu práce uvádíme základy z teorie Petriho sítí. V další části práce popisujeme metody z teorie grafů používané při analýze Petriho sítí a jejich aplikaci při testování vlastností Petriho sítí. Dále se podrobněji věnujeme grafovým technikám používaným při analýze neomezených Petriho sítí. Další kapitola je věnovaná popisu přidaných vizualizačních možností nástroje. V závěru pak uvádíme ukázky implementovaných vylepšení a analytických metod.

Klíčová slova: diplomová práce, Petriho sítě, teorie grafů, New Modified Reachability Tree, automatický layout

Abstract

In this master thesis we describe the extensions and modifications of the tool for editing and analyzing of P/T Petri nets that is being developed at the department of computer science. The tool consists of web graphical editor, analysis server and database storage. We describe the fundamentals of Petri nets theory in the introduction. Then we describe methods of graph theory that can be used for Petri net analysis and their application in Petri net property tests. In the next chapter the methods of analyzing unbounded Petri nets are introduced. After that we represent the graphical extensions of the editor. At the end we show the examples of the implemented extensions and analysis methods.

Key Words: master thesis, Petri nets, graph theory, New Modified Reachability Tree, automatic layout

Obsah

Seznam použitých zkratek a symbolů

ART	–	Augmented Reachability Tree
DFS	–	Depth-First Search - Prohledávání do hloubky
FRT	–	Finite Reachability Tree
JSON	–	JavaScript Object Notation
MRT	–	Modified Reachability Tree
NMRT	–	New Modified Reachability Tree
PN	–	Petri Nets - Petriho sítě
RG	–	Reachability Graph - Graf dosažitelnosti
RS	–	Reachability Set - Množina dosažitelnosti
RT	–	Reachability Tree - Strom dosažitelnosti

Seznam obrázků

1 Úvod

Petriho sítě jsou užitečným nástrojem pro modelování a analýzu paralelních a distribuovaných systémů. V současnosti existuje řada tříd Petriho sítí s různými možnostmi a schopnostmi zachycení informací modelovaných systémů. Cílem této diplomové práce je rozšířit již existující nástroj pro editaci a analýzu Petriho sítí, který je vyvíjen na katedře informatiky a zaměřuje se na třídu Place/Transition Petriho sítí bez kapacity míst. Rozšíření tohoto nástroje v rámci této diplomové práce spočívá v přidání nových analytických metod, v rozšíření množiny analyzovatelných vlastností Place/Transition Petriho sítí a také v rozšíření vizualizačních možností. Hlavním zaměřením této práce je rozšířit existující nástroj o analytické možnosti využívající techniky z teorie grafů. Týká se to jak nových analytických metod, tak i nových analyzovatelných vlastností, při jejichž analýze jsou uplatňovány algoritmy z teorie grafů.

Prvním důležitým bodem této práce bylo seznámení se s existujícím nástrojem a zároveň jeho nasazení na server, aby mohl být využíván při výuce. Jedná se o komplexní nástroj, jehož součástí je editor ve formě webového klienta, databázové úložiště a analytický server. Autor při jeho tvorbě použil řadu různých technologií, programovacích jazyků a frameworků, což značně ztížilo pochopení jeho práce a seznámení se s nástrojem. Jelikož s některými použitými technologiemi jsem před psaním této práce neměl žádné zkušenosti, musel jsem dostudovat jejich možnosti, fungování a syntaxi.

Začátek práce je věnován stručnému popisu základů teorie P/T Petriho sítí. Jsou zde uvedeny potřebné definice, pravidla chování a také vlastnosti Petriho sítí. Popisujeme zde základní vlastnosti, jejichž testování se v rámci analýzy Petriho sítí provádí.

V kapitole 3 uvádíme metody z teorie grafů, které lze aplikovat při analýze Petriho sítí na testování vlastností uvedených v kapitole 2. Zaměřujeme se zde zejména na způsoby testování vlastností omezených Petriho sítí a v závěru kapitoly pak popisujeme nejběžnější metodu analýzy neomezených sítí.

Analýze neomezených sítí se více věnujeme v kapitole 4, kde popisujeme generování New Modified Reachability Tree a jeho možné aplikace při analýze především neomezených Petriho sítí. Uvádíme zde také samotný algoritmus a způsob implementace do existujícího programu.

Kapitola 5 je věnována popisu rozšíření vizualizačních možností stávajícího programu. Popisujeme zde řešení ukládání stavů editoru a také nový automatický layout, který může uživatel využít k vykreslení manuálně vytvořené Petriho sítě.

V závěru práce pak uvádíme ukázky nově implementovaných možností programu. Názorně zde předvádíme použití nového automatického layoutu na modelovanou Petriho síť. Dále na příkladu demonstrujeme přínos New Modified Reachability Tree pro analýzu neomezených Petriho sítí.

2 Teoretický úvod o Petriho sítích

Petriho sítě jsou formálním a grafickým jazykem vhodným pro modelování distribuovaných a paralelních systémů, který umožňuje jejich popis pomocí grafické a matematické reprezentace. Matematická reprezentace slouží k určení vlastností a simulaci chování, zatímco grafická reprezentace umožňuje znázornění procesů v systému. Petriho sítě vznikly jako zobecnění teorie automatů.

Poprvé byly představeny v roce 1962 Carlem Adamem Petrim v jeho dizertační práci a od té doby jsou stále vyvíjeny, rozšiřovány a aplikovány v různých oblastech. Postupně vznikla řada tříd Petriho sítí a také mnoho nástrojů, které s nimi pracují, umožňují jejich grafické znázornění, modelování či analýzu. Výzkum a vývoj Petriho sítí intenzivně probíhá i v současné době.

Petriho sítě byly rychle přijaty jako jeden z nejadekvátnějších jazyků pro popis a analýzu komunikace a sdílení zdrojů mezi souběžnými procesy. Příkladem jejich využití je modelování chování komunikačních protokolů nebo výrobních procesů. V práci [24] je uveden přehled aplikací Petriho sítí. Postupně pak byly Petriho sítě obohacovány a upravovány tak, aby byly více použitelné v praxi, a aby tak více odpovídaly reálným praktickým požadavkům. Důsledkem byl vznik různých tříd Petriho sítí, mezi které například patří:

- P/T (Place/Transition) Petriho sítě,
- P/T Petriho sítě s inhibičními hranami,
- P/T Petriho sítě s prioritami.

Více o zmíněných jednotlivých třídách Petriho sítí lze nalézt ve skriptech [11], ze kterých budeme také vycházet v této kapitole při zavádění definic a pojmů z oblasti Petriho sítí, nebude-li uveden jiný zdroj.

Tato práce se zaměřuje pouze na první z výše jmenovaných tříd, a sice třídu P/T Petriho sítí bez inhibičních hran a priorit. Na tuto třídu je také cílen nástroj rozšiřovaný v rámci této práce. V této kapitole uvedeme základní teorii týkající se třídy P/T Petriho sítí a v této i v následujících kapitolách bude pojmem Petriho sítě vždy myšlena tato třída, nebude-li řečeno jinak.

2.1 Struktura Petriho sítě

Petriho síť, konkrétně její strukturu, tvoří několik různých komponent. První z nich jsou místa. Místa v Petriho síti reprezentují její aktuální stav, ovšem nemají schopnost aktivně tento stav změnit. Slouží tedy k uchovávání informace o stavu sítě, respektive zkoumaného systému znázorněného touto Petriho sítí. Každé místo za tímto účelem obsahuje *tokeny*, které představují konkrétní informaci o stavu sítě. Další komponentou jsou přechody. Přechody reprezentují možné akce v rámci modelovaného systému, které umožňují měnit aktuální stav. V rámci Petriho sítí mají schopnost vytvářet, konzumovat nebo přemisťovat tokeny mezi místy, čímž aktivně mění

stav sítě. Vzájemné vztahy mezi místy a přechody sítě jsou reprezentovány další komponentou, a to hranami. Hrany mají podobu orientovaných šipek a jsou grafickým znázorněním propojení míst a přechodů. Protože hrana může spojovat pouze místo a přechod, nikoli dvě místa nebo dva přechody, jedná se o bipartitní graf.

Součástí struktury Petriho sítě jsou také kapacity míst a ohodnocení hran. Každé místo má svou kapacitu, což je maximální možná hodnota počtu tokenů, které se v jednu chvíli mohou nalézat v daném místě. Není-li uvedena žádná hodnota, implicitně předpokládáme hodnotu kapacity ∞. Ohodnocení, nebo také váha hrany, je přirozené číslo, udávající násobnost hrany. Ta určuje, kolik tokenů se po této hraně přesunuje. Není-li uvedena žádná hodnota, předpokládáme násobnost hodnoty 1.

Definice 1 *Struktura Petriho sítě je čtveřice* $\langle P, T, I, O \rangle$*, ve které*

- P *značí konečnou množinu míst,*
- T *značí konečnou množinu přechodů,*
- I *značí tzv. vstupní funkci, což je zobrazení typu* $T \to P_{MS}$*,*
- O *značí tzv. výstupní funkci, což je také zobrazení typu* $T \to P_{MS}$*,*
- P_{MS} *je množina všech multimnožin nad množinou* P*.*

Tato definice se mírně liší od definice uvedené v [11], kde je P/T Petriho síť definována jako pětice obsahující navíc ještě zobrazení H pro vstupní inhibiční funkce, ovšem cílem této práce jsou obyčejné P/T Petriho sítě bez inhibičních hran, a proto tomu budou přizpůsobeny i definice.

K zobrazení struktury Petriho sítě se používá diagram, ve kterém jsou místa znázorněna kružnicí, přechody jsou znázorněny obdélníky (někdy se pro zjednodušení používá pouze úsečka). Vstupní funkce jsou znázorněny pomocí šipek, vedoucích z míst do přechodů a výstupní funkce naopak pomocí šipek, vedoucích z přechodů do míst.

2.2 Systém Petriho sítě

Systém Petriho sítě je struktura Petriho sítě, ke které navíc přidáme informace o jejím stavu reprezentované tokeny v místech této sítě, což se nazývá značení Petriho sítě. To lze formálně zapsat jako zobrazení $P \to \mathbb{N}$, kde $\mathbb{N}$ představuje množinu všech nezáporných celých čísel. Jedná se tedy o zobrazení z množiny míst do množiny přirozených čísel. Pro zápis počtu tokenů v daném místě p se používá značení M(p). Počet tokenů v daném značení M pro jednotlivá místa lze zapsat jako vektor $M = (m_1, m_2, \ldots, m_i)$, kde m_j představuje počet tokenů v místě p_j pro $j = 1, 2, 3, \ldots, |P|$. Dále platí, že $m_j = M(P_j)$. Vektoru M se také říká *vektor značení.*

Definice 2 *Systém Petriho sítě je pětice* $\langle P, T, I, O, M_0 \rangle$*, ve které*

- $\langle P, T, I, O \rangle$ *představuje strukturu z definice 1,*

- *M_0 je tzv. počáteční značení, udávající počet tokenů v jednotlivých místech v počátečním stavu sítě.*

Počet tokenů v diagramu se znázorňuje tečkami uvnitř kružnice reprezentující dané místo, pokud je počet tokenů malý, případně číslem vepsaným do této kružnice, je-li počet tokenů větší a použití teček by nebylo přehledné.

Speciálním případem počátečního značení může být situace, kdy pro všechna $p \in P$ platí, že $M_0(p) = 0$. V takovém případě systém Petriho sítě zachycuje pouze její strukturu.

Definice 3 *Pro popis Petriho sítě se dále používají tato značení:*

- *I(t) - multimnožina vstupních míst přechodu t,*
- *O(t) - multimnožina výstupních míst přechodu t,*
- *I(t,p) - násobnost hrany z místa p do přechodu t,*
- *O(t,p) - násobnost hrany z přechodu t do místa p,*
- $^\bullet t = \{p \in P : I(t,p) > 0\}$ *- množina vstupních míst přechodu t,*
- $t^\bullet = \{p \in P : O(t,p) > 0\}$ *- množina výstupních míst přechodu t,*
- $^\bullet p = \{t \in T : O(t,p) > 0\}$ *- množina vstupních přechodů místa p,*
- $p^\bullet = \{t \in T : I(t,p) > 0\}$ *- množina výstupních přechodů místa p.*

2.3 Dynamika Petriho sítě

Dynamika Petriho sítí je definována pravidly pro změny stavu sítě. Možnost dynamicky měnit stav sítě umožňuje simulaci chování modelovaného systému. Jak již bylo řečeno, Petriho síť pomocí míst a tokenů zaznamenává aktuální stav modelovaného systému. Stav sítě je tedy určen jejím značením a může se měnit provedením akce, což je reprezentováno *odpálením* nebo také *provedením* přechodu. Stav Petriho sítě se může měnit podle následujících pravidel:

- Vstupní místa přechodu t jsou taková, ze kterých vede hrana do tohoto přechodu,
- Výstupní místa přechodu t jsou taková, do kterých vede hrana z tohoto přechodu,
- Přechod je proveditelný, pokud je počet tokenů v každém vstupním místě p přechodu t větší nebo roven násobnosti hrany vedoucí z místa p do přechodu t,
- Provedením přechodu se změní stav sítě tak, že v každém jeho vstupním místě se počet tokenů sníží o hodnotu násobnosti příslušné spojující hrany a v každém výstupním místě se počet tokenů naopak zvýší o hodnotu násobnosti příslušné spojující hrany.

Dynamika Petriho sítě zejména popisuje podmínku proveditelnosti přechodu a pravidlo provedení přechodu.

Proveditelnost přechodu Přechod je proveditelný, pokud každé z jeho vstupních míst obsahuje alespoň tolik tokenů, kolik je váha příslušné hrany vedoucí z daného místa do přechodu. Je-li přechod proveditelný, může být proveden.

Definice 4 *Přechod t je proveditelný ve značení M, pokud platí* $(\forall p \in {}^\bullet t)[M(p) \geq I(t,p)]$.

Množina všech proveditelných přechodů ve značení M se značí E(M). Není-li proveditelný žádný přechod, platí $E(M) = \emptyset$. Značení, ve kterém není proveditelný žádný přechod, se nazývá uzamčení (neboli deadlock).

Protože v původním programu byla definice P/T Petriho sítí modifikována podle [11] tak, že kapacita míst je rovna ∞, budeme v této práci používat stejnou modifikaci. Neuvažování kapacity míst usnadňuje analýzu a nijak přitom nesnižuje modelovací schopnosti Petriho sítí (viz kapitola 2, str. 1 v [11]).

Provedení přechodu Provedení přechodu reprezentuje akci v modelovaném systému. Výsledkem provedení přechodu proto zpravidla bývá změna aktuálního stavu. Samotné provedení přechodu odebere určitý počet tokenů ze všech vstupních míst přechodu a naopak zvýší počet tokenů ve všech výstupních místech. Počet tokenů přidaných do výstupních míst přitom nezávisí na počtu tokenů spotřebovaných ve vstupních místech. V obou případech počet odebraných i přidaných tokenů závisí na násobnosti příslušné hrany spojující místo s přechodem. Z definice 4 a z pravidla, že proveden může být vždy pouze proveditelný přechod, také vyplývá, že z místa nemůže být nikdy odebráno více tokenů, než kolik se jich v něm nachází.

Provádění přechodů v Petriho síti je nedeterministické, což znamená, že pokud je proveditelných více přechodů ve značení M, nedeterministicky je zvolen jeden z těchto přechodů, který se provede.

Nachází-li se síť ve značení M, potom se provedením přechodu t dostane do značení M', což lze zapsat jako $M \xrightarrow{t} M'$. Pro značení M' platí, že je *bezprostředně dosažitelné* ze značení M. Pro M' také platí, že $M' = M + O(t) - I(t)$, neboli

$$(\forall p \in P)[M'(p) = M(p) + O(t,p) - I(t,p)]$$

.

T. Murata ve své práci [13] uvádí dva speciální typy přechodů, které nazývá *source* a *sink*, neboli *generátor* a *konzument* (někdy se používají i výrazy *zdroj* a *spotřebič*). Generátor tokenů je takový přechod, který nemá žádné vstupní místo, tedy platí pro něj (${}^\bullet t = \emptyset$). Z toho vyplývá, že podle definice 4 je tento přechod vždy proveditelný v libovolném značení. Proto také platí, že pokud Petriho síť obsahuje takovýto generátor tokenů, pak neobsahuje uzamčení. Při simulování se tento přechod chová podle svého názvu jako generátor a může kdykoli zvyšovat počet tokenů v síti generováním tokenů do svých výstupních míst. Konzument tokenů je přechod, který nemá

žádná výstupní místa, pouze vstupní, takže pro něj platí ($t^\bullet = \emptyset$). Takovýto přechod v síti odebírá tokeny, ale žádné negeneruje, takže jeho provedením vždy dojde ke snížení počtu tokenů v síti.

Posloupnost přechodů Jedná se o zobecnění pojmu provedení přechodu. Posloupnost přechodů reprezentuje množinu přechodů, které lze postupně provést v daném pořadí, a značíme ji $\sigma = t_{(1)}, t_{(2)}, \ldots, t_{(n)}$. Provedením dané posloupnosti přechodů se síť dostane ze značení M do značení M', což značíme $M \xrightarrow{\sigma} M'$. Pokud existuje taková posloupnost σ, pro kterou platí $M \xrightarrow{\sigma} M'$, pak lze říci, že značení M' je dosažitelné ze značení M.

Množina dosažitelnosti Podle definice 2 je systém Petriho sítě definován jako pětice, obsahující počáteční značení M_0, které reprezentuje počáteční stav systému. Při analýze nás pak zajímají ta značení, která jsou dosažitelná právě z počátečního značení. Množina těchto dosažitelných značení se nazývá *množina dosažitelnosti* a značí se $RS(M_0)$, případně pouze RS (Reachability Set).

Definice 5 *Množina dosažitelnosti $RS(M_0)$ je definovaná takto:*

- $M_0 \in RS(M_0)$,
- $M \in RS(M_0) \wedge (\exists t \in T)[M \xrightarrow{t} M'] \Rightarrow M' \in RS(M_0)$.

Množina dosažitelnosti je také základem pro vytvoření stromu, respektive grafu dosažitelnosti, což je jedno ze základních východisek při analýze Petriho sítí. Strom dosažitelnosti, respektive její graf, je grafovou reprezentací množiny dosažitelnosti spolu s relací dosažitelnosti. Algoritmus konstrukce množiny dosažitelnosti vypadá takto:

Input: (PN, M_0)
1 Inicializuj množinu $X = \{M_0\}$;
2 **do**
3 | Do množiny X přidáme všechna značení bezprostředně dosažitelná ze značení, která již patří do X;
4 **while** *došlo ke zvětšení množiny* X;

Algoritmus 1: Algoritmus konstrukce množiny dosažitelnosti.

Z algoritmu konstrukce množiny dosažitelnosti a z existence generátorů tokenů také vyplývá možnost, že množina dosažitelnosti bude nekonečná. Existuje-li totiž v síti generátor tokenů, může neomezeně zvyšovat počet tokenů ve svých výstupních místech, čímž bude vytvářet nová značení. V takovém případě bude nekonečný také strom dosažitelnosti dané sítě, což je komplikace pro analýzu. Z tohoto důvodu jsou způsoby analýzy Petriho sítí s nekonečnou množinou dosažitelnosti stále předmětem aktivního výzkumu, který má za cíl vytvořit způsob konečné reprezentace nekonečné množiny s co nejmenší ztrátou informací. Některým z těchto metod se budeme dále věnovat v pozdější části této práce.

Graf dosažitelnosti Množina vrcholů stromu dosažitelnosti, případně grafu dosažitelnosti (RG - Reachability Graph), je tvořena množinou dosažitelných značení Petriho sítě. Jeho hrany jsou ohodnoceny přechody, jimiž se mezi danými značeními přechází. Díky tomu, že zachycuje možná značení sítě, je velmi užitečným pomocníkem pro analýzu jejích vlastností.

Definice 6 *Pro Petriho síť s množinou dosažitelnosti $RS(M_0)$ je $RG(M_0)$ její graf dosažitelnosti, což je hranově ohodnocený graf s těmito vlastnostmi:*

- *$RS(M_0)$ je množinou uzlů grafu.*
- *Pro množinu hran A platí:*
  - $A \subseteq RS \times RS \times T$,
  - $\langle M_i, M_j, t \rangle \in A \Leftrightarrow M_i \xrightarrow{t} M_j$,
- *M_0 je počáteční (kořenový) uzel grafu.*

2.4 Vlastnosti Petriho sítí

Při analýze Petriho sítě se pomocí analytických metod testují vlastnosti sítě a tím i systému, který je touto sítí reprezentován. Množina vlastností Petriho sítí je široká, a proto zde popisovaný analytický nástroj umožňuje analýzu pouze některých vlastností z této množiny. V této části práce budou uvedeny základní vlastnosti Petriho sítí a především ty vlastnosti, jejichž analýzou se budeme zabývat v rámci této práce.

2.4.1 Živost

Petriho síť je živá, jestliže každý přechod sítě může být proveden v některém z dosažitelných značení bez ohledu na to, v jakém značení se síť právě nachází. To tedy znamená, že každý z přechodů se může stát znovu proveditelným bez ohledu na zvolenou posloupnost přechodů.

Definice 7 *Přechod je živý ve značení M, pokud platí:*
$(\forall M' \in RS(M))(\exists M'' \in RS(M'))[t \in E(M'')]$.

Petriho síť je živá, pokud platí:
$(\forall t \in T)(\forall M \in RS(M_0))(\exists M' \in RS(M))[t \in E(M')]$.

2.4.2 Omezenost

Omezenost místa značí maximální počet tokenů, který se v daném místě vyskytuje, ze všech dosažitelných značení. Omezenost Petriho sítě udává maximální počet tokenů v libovolném místě sítě ze všech dosažitelných značení. U neomezených sítí s nekonečnou množinou dosažitelnosti je tato hodnota nekonečná. Míra omezenosti se obvykle označuje k.

Definice 8 *Místo* p *v PN-systému je k-omezené, pokud je v tomto místě maximální dosažitelný počet tokenů v libovolném dosažitelném značení nanejvýše roven k, tedy platí:*
$(\forall M \in RS(M_0))[M(p) \leq k]$.

PN-systém je k-omezený, pokud jsou všechna jeho místa k-omezená, tedy platí:
$(\forall p \in P)(\forall M \in RS(M_0))[M(p) \leq k]$.

Důležitým důsledkem omezenosti je fakt, že daná Petriho síť má konečnou množinu dosažitelnosti, což je velmi důležitá informace pro její další analýzu.

Bezpečnost Bezpečnost je vlastně speciální případ omezenosti, kdy platí, že každé místo, a tedy i celá Petriho síť, je *1-omezená*.

2.4.3 Reverzibilita

Reverzibilita, nebo také reverzibilnost, Petriho sítě značí její schopnost vrátit se z jakéhokoli značení do značení počátečního. Petriho síť je tedy reverzibilní, pokud z každého jejího dosažitelného značení existuje posloupnost přechodů, kterou se síť dostane do počátečního značení.

Definice 9 *Značení je vždy dosažitelné, pokud je dosažitelné z každého dosažitelného značení, tedy platí:*
$(\forall M' \in RS(M_0)[M \in RS(M')])$.

PN-systém je reverzibilní, pokud je jeho počáteční značení vždy dosažitelné, tedy platí:
$(\forall M \in RS(M_0))[M_0 \in RS(M)]$.

Problém dosažitelnosti Jedná se o jeden z klasických problémů analýzy Petriho sítě. Otázka zní, zda dané značení je dosažitelné ze značení počátečního, tedy zda se může modelovaný systém dostat do nějakého konkrétního stavu. Řešení tohoto problému je komplikované především při analýze neomezené sítě s nekonečnou množinou dosažitelnosti a je to jeden z bodů, na který se zaměřuje rozšíření analytického programu v rámci této práce.

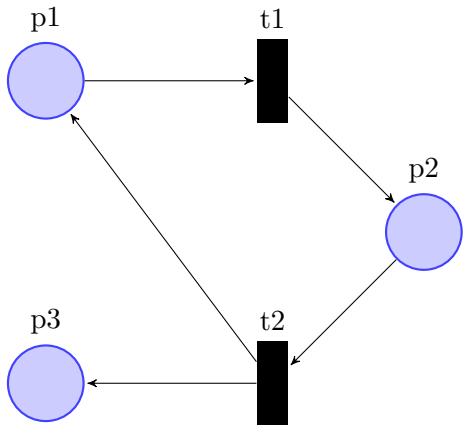
2.4.4 Uzamčení

Uzamčení je stav Petriho sítě, ve kterém není proveditelný žádný přechod. V [11] se uvádí, že uzamčení může, ale také nemusí, být žádoucí. Záleží na požadavcích kladených na systém modelovaný danou Petriho sítí. Pro některé systémy může být dosažitelnost uzamčení nutností, pro jiné naopak nepřijatelnou situací.

Z hlediska analýzy Petriho sítě je důležitý fakt, že přítomnost uzamčení jasně určuje, že síť není živá a také že není reverzibilní, protože z uzamčení se nelze dostat do žádného jiného značení, a tedy ani do počátečního. Zároveň ovšem na základě nepřítomnosti uzamčení nelze říci, že síť je živá nebo že je reverzibilní. Obdobně je tomu i naopak, tedy že je-li síť neživá nebo nereverzibilní, neznamená to existenci uzamčení.

2.4.5 Repetičnost

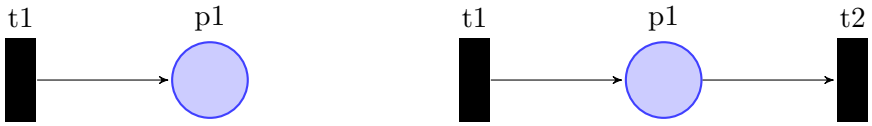
Podle [13] je Petriho síť repetiční, pokud v ní existuje počáteční značení M_0 a sekvence přechodů σ začínající v M_0 taková, že obsahuje každý přechod sítě libovolně mnohokrát. Pokud tato sekvence obsahuje pouze některé přechody, pak se nazývá *částečně* repetiční. Pokud sekvence σ obsahuje každý přechod sítě právě jednou, je tato síť *striktně* repetiční. Příklad repetiční sítě je na obrázku 1. Tato síť je strukturálně repetiční, a pokud bychom umístili token do místa *p1*, byla by repetiční.



Obrázek 1: Strukturálně repetiční, ovšem nikoliv konzistentní, Petriho síť (převzato z [13]).

2.4.6 Konzistenčnost

Jedná se o speciální případ repetičnosti. Petriho síť je konzistentní, pokud existuje značení M_0 a sekvence přechodů σ vedoucí z M_0 zpět do M_0 taková, že obsahuje každý přechod sítě alespoň jednou. Síť je částečně konzistentní, pokud tato sekvence σ obsahuje pouze některé přechody alespoň jednou.



Obrázek 2: Síť vlevo je repetiční, síť vpravo je repetiční i konzistentní (převzato z [13]).

Na obrázku 2 jsou zobrazeny dvě Petriho sítě, které demonstrují rozdíl mezi konzistenčností a repetičností. Síť vlevo není konzistentní, protože neexistuje sekvence přechodů σ taková, která obsahuje alespoň jedno provedení přechodu a zároveň vede znovu do počátečního značení.

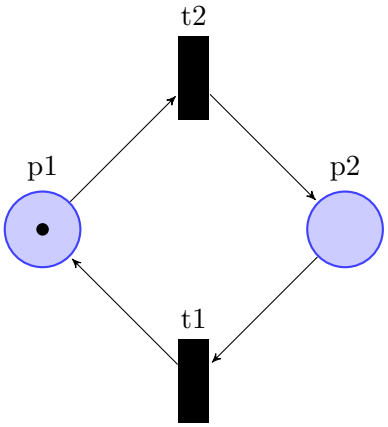
2.4.7 Konzervativnost

Konzervativnost říká, že v síti nezanikají ani nevznikají tokeny. To může být žádoucí v situacích, kdy tokeny reprezentují nějaké dostupné zdroje, jejichž počet je konstantní. V takovém případě se samozřejmě počet tokenů nesmí měnit. Vzhledem k tomu, že takovému zdroji nemusí vždy odpovídat právě jeden token, definuje se konzervativnost pomocí váhy tokenů, někdy zapisované formou váhového vektoru $w = (w_1, w_2, \ldots, w_n),\ n = |P|$ *a* $w_i \geq 0$ *pro* $i = 1, 2, \ldots, n$.

Definice 10 *Podle [13] je Petriho síť konzervativní, pokud existuje kladné celé číslo y(p) pro každé místo p takové, že váha tokenů* $M^T y = M_0^T y =$ *konstanta, pro* $\forall M \in RS(M_0)$.

Stejně jako je tomu u repetičnosti, i u konzervativnosti platí, že síť může být pouze částečně konzervativní nebo naopak striktně konzervativní. Pokud není definice 10 splněna pro každé místo p, ale pouze pro některá místa, pak se síť nazývá částečně konzervativní. Pokud je váha w, neboli hodnota $y(p)$, pro každé místo rovna 1, pak je síť striktně konzervativní.

Příklad konzervativní sítě je na obrázku 3. Tato síť je mimo jiné také repetiční i konzistentní.



Obrázek 3: Ukázka konzervativní Petriho sítě (převzato z [13]).

3 Grafové metody pro analýzu Petriho sítí

V práci [13] se uvádí, že existují tři přístupy k analýze Petriho sítí. Prvním je přístup založený na analýze stromu dosažitelnosti či stromu pokrytí. Taková analýza obvykle vyžaduje množinu všech dosažitelných značení, což je komplikace v případě analýzy neomezené sítě, která má nekonečnou množinu dosažitelnosti. Výhodou tohoto přístupu je fakt, že je aplikovatelný na všechny třídy Petriho sítí. Nevýhoda pak spočívá v tom, že je tento přístup aplikovatelný jen na sítě s „malým“ stromem dosažitelnosti kvůli *explozi stavového prostoru*, kdy může počet dosažitelných stavů dosahovat obrovských čísel a analýza pak může být paměťově i časově extrémně náročná i při dnešních technologických možnostech. Analýza stromu dosažitelnosti či pokrytí se obvykle využívá pro detekci vlastností chování Petriho sítě, jako jsou například živost či reverzibilita.

Druhým přístupem je analýza využívající matici incidence Petriho sítě. Tímto způsobem se detekují strukturální vlastnosti Petriho sítě, protože nepracuje s počátečním značením, ale pouze se strukturou Petriho sítě. Typicky se jedná o rovnici (nerovnici) obsahující incidenční matici a hledání vektoru, po jehož dosazení bude rovnice (nerovnice) platná. Tato metoda je účinná, ovšem podle [13] je její využití omezené. Lze ji totiž využít pouze pro analýzu některých podtříd Petriho sítí, případně ve speciálních situacích.

Třetím přístupem jsou redukční nebo dekompoziční techniky. Pro ně však platí totéž, co pro druhý přístup - jedná se o silnou metodu, ovšem s omezeným využitím.

Analýzu Petriho sítí lze rozdělit na dvě varianty na základě omezenosti. Analýza omezených sítí je výrazně jednodušší než analýza sítí neomezených. Proto se budeme analytickým metodám věnovat pro každou variantu zvlášť.

3.1 Pojmy z teorie grafů

Pro popis algoritmů a metod analýzy v této kapitole je potřeba nejdříve zavést některé základní pojmy z teorie grafů. Zde uvedené definice jsou převzaty z [8].

Definice 11 *Jednoduchý graf G je uspořádaná dvojice (V,E), kde V je neprázdná množina vrcholů a E je nějaká množina dvouprvkových podmnožin množiny V. Prvkům E říkáme hrany.*

V takovém grafu hrany nemají žádnou orientaci. Petriho síť je orientovaný graf, ve kterém jsou hrany v podobě šipek.

Definice 12 *Orientovaný graf G je dvojice (V,A), kde V je neprázdná množina vrcholů a A je podmnožina kartézského součinu $V \times V$ (množina uspořádaných dvojic prvků z V. Prvky z množiny A se nazývají orientované hrany.*

Přestože Petriho sítě neobsahují žádné smyčky, jejich hrany mají násobnost, takže se nejedná o jednoduchý orientovaný graf.

Definice 13 *Sled v grafu G je taková posloupnost vrcholů a hran* $(v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n)$*, ve které každá orientovaná hrana* e_i *má počáteční vrchol* v_{i-1} *a koncový vrchol* v_i *pro* $i = 1, 2, \ldots, n$*. Pokud* $v_0 = v_n$*, potom se jedná o uzavřený sled.*

Sled lze chápat jako záznam procházení grafem. Hrany i vrcholy se mohou libovolně opakovat.

Definice 14 *Tah je sled, ve kterém se žádná hrana neopakuje. (u,v)-tah je tah s počátečním vrcholem u a koncovým vrcholem v. Pokud u = v, potom se jedná o uzavřený tah.*

Definice 15 *Cesta je sled, ve kterém se žádný vrchol neopakuje. Důsledek toho je, že se neopakuje ani žádná hrana. (u,v)-cesta je cesta s počátečním vrcholem u a koncovým vrcholem v.*

Definice 16 *Cyklus v grafu G je uzavřená cesta. V uzavřené cestě je počáteční vrchol zároveň i koncovým, což je výjimka z její definice, že se žádný vrchol neopakuje.*

Definice 17 *Orientovaný sled v orientovaném grafu G je taková posloupnost vrcholů a hran* $(v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n)$*, ve které každá orientovaná hrana* e_i *má počáteční vrchol* v_{i-1} *a koncový vrchol* v_i *pro* $i = 1, 2, \ldots, n$*. Orientovaný sled, ve kterém se neopakuje žádná hrana, se nazývá orientovaný tah. Orientovaný sled, ve kterém se neopakuje žádný vrchol, se nazývá orientovaná cesta. Definice orientovaného sledu, tahu a cesty jsou analogické k definicím obecného sledu, tahu a cesty.*

3.2 Analýza omezené Petriho sítě

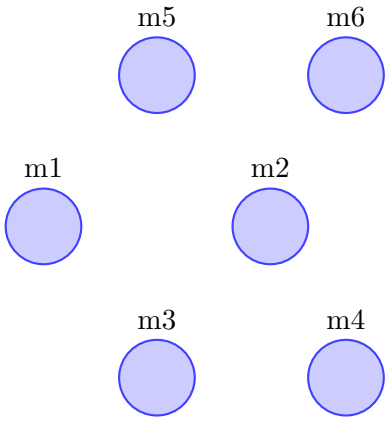
Je-li Petriho síť omezená, znamená to, že má konečný počet dosažitelných značení. V takovém případě jsou možnosti její analýzy větší, než je tomu v případě neomezené sítě, protože máme, alespoň teoreticky, k dispozici všechna dosažitelná značení. Jak je uvedeno v [11], v praxi se může stát, že množina dosažitelnosti $RS(M_0)$ definovaná v definici 5 je tak početná, že časová i paměťová náročnost analýzy celé množiny přesáhne hranici přijatelnosti. Pokud však není $RS(M_0)$ příliš rozsáhlá, lze její analýzou získat informace o vlastnostech chování příslušné Petriho sítě. Za tímto účelem se konstruuje *graf dosažitelnosti (RG)*, případně *strom dosažitelnosti (RT)*.

3.2.1 Generování grafu dosažitelnosti

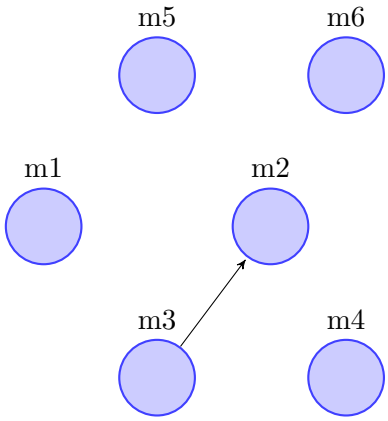
Graf dosažitelnosti lze v zásadě generovat dvěma různými způsoby. Oba způsoby jsou podobné a vedou ke stejnému výsledku.

První způsob lze použít, pokud již známe množinu dosažitelnosti Petriho sítě. V takovém případě nejdříve vytvoříme pro každý jedinečný prvek množiny dosažitelnosti jeden uzel v grafu. Každé jedinečné dosažitelné značení tak bude reprezentováno právě jedním uzlem. V dalším kroku pak pro každé dosažitelné značení nalezneme všechny proveditelné přechody, pro každý z nich zjistíme, do jakého značení se jeho provedením síť dostane, a následně tuto informaci

zaneseme do grafu. Zanesením do grafu je myšleno vytvoření orientované hrany z výchozího uzlu do uzlu reprezentujícího značení dosažené provedením přechodu, jehož označení použijeme pro ohodnocení této hrany. Názorná ukázka je na obrázcích 4 a 5. Na obrázku 4 jsou uzly grafu. Řekněme, že zjistíme, že ve značení *m3* je proveditelný přechod *t2* a že jeho provedením se síť dostane do značení *m2*. Zanesení této informace je znázorněno na obrázku 5.



Obrázek 4: Vrcholy grafu dosažitelnosti.



Obrázek 5: Přidání hrany z m3 do m2.

Tento způsob však obvykle není příliš praktický, především pro ruční tvorbu grafu, a to hlavně ze dvou důvodů. Protože obvykle známe pouze strukturu sítě a její počáteční značení, museli bychom nejdřív vygenerovat množinu dosažitelnosti pro daný *PN-systém*. Při generování bychom však nutně museli pro jednotlivá značení zjišťovat jejich proveditelné přechody a nová dosažitelná značení bychom objevovali zjišťováním, do kterého značení se dostaneme provedením přechodů. Je zjevné, že bychom tak tyto kroky dělali vlastně dvakrát - při generování množiny dosažitelnosti a při tvorbě grafu dosažitelnosti. Druhým důvodem je skutečnost, že nelze předem odhadnout, která značení budou spojena hranou, a proto může být výsledný graf velmi nepřehledný kvůli častému křížení hran či velké vzdálenosti hranou spojených uzlů.

Druhým způsobem je generování grafu dosažitelnosti pomocí algoritmu, vycházejícího z *prohledávání do šířky*.

Prohledávání grafu do šířky Algoritmus prohledávání grafu do šířky (anglicky breadth-first search) je algoritmus vhodný především pro hledání nejkratších cest v grafu a jedná se o jeden ze dvou základních a nejčastěji používaných grafových algoritmů. Podle [16] a [10] byl poprvé použit v práci [12], kde ho E. F. Moore používal pro hledání nejkratších cest z bludiště. Nezávisle na něm ho objevil také C. Y. Lee při hledání nejlepších cest, což popsal v [9]. Podrobně je algoritmus popsán například v [16]. Běh algoritmu pro graf G a počáteční vrchol s vypadá

takto:

```
Input: G, s
Všechny uzly grafu, kromě s, označ jako FRESH a bez rodiče;
Kořenový uzel označ jako bez rodiče a OPEN;
Vytvoř frontu Q = {s};
while fronta je neprázdná do
    u = dequeue[Q] (dequeue - odebrání prvního prvku fronty);
    Zpracuj uzel u;
    for každý vrchol v, který je potomkem u, do
        Zpracuj hranu (u, v);
        if v je FRESH then
            Označ v jako OPEN;
            Nastav u jako rodiče v;
            Přidej v do Q;
        end
    end
    Označ u jako CLOSED;
end
```

Algoritmus 2: Prohledávání grafu do šířky.

Algoritmus prohledávání do šířky tedy začne v kořenovém uzlu a ve vlnách projde postupně všechny uzly grafu. Nejdříve uzly přímo sousedící s kořenem, potom uzly se vzdáleností 2 od kořene, pak uzly se vzdáleností 3 od kořene a takto pokračuje, dokud neprojde všechny uzly grafu.

Algoritmus generování grafu dosažitelnosti Algoritmus generování grafu dosažitelnosti funguje podobně. Začne v kořenovém uzlu, který reprezentuje počáteční značení a je v tu chvíli jediným uzlem grafu. Najde všechna značení, dosažitelná z počátečního, vytvoří odpovídající uzly grafu i příslušné hrany, jejichž ohodnoceními jsou názvy odpovídajících přechodů. Poté všechny nově přidané uzly postupně zpracuje stejným způsobem, jakým byl zpracován kořenový uzel. Takto algoritmus pokračuje, dokud v grafu existují dosud nezpracované uzly. Výsledkem je graf dosažitelnosti a zároveň i množina dosažitelnosti. Výhodou tohoto postupu je to, že graf vzniká postupně ve vlnách a je tedy obvykle poměrně přehledný bez velkého počtu křížení hran.

Jinou variantou grafu dosažitelnosti je strom dosažitelnosti, což je vlastně acyklický graf dosažitelnosti. Ve stromu dosažitelnosti nejsou žádné cykly a zpětné hrany, které mohou být v grafu dosažitelnosti, a proto se rozvíjí pouze jedním směrem, kterým také vedou všechny hrany. Generování stromu dosažitelnosti je velmi podobné generování grafu dosažitelnosti, ovšem místo přidání zpětné hrany je přidán duplicitní uzel, který se už ovšem nebude zpracovávat. Výsledkem je tak přehlednější graf, jehož nevýhodou je ale větší počet uzlů v grafu kvůli existenci duplicitních uzlů.

Množina dosažitelnosti $RS(M_0)$, případně její grafová reprezentace v podobě grafu či stromu dosažitelnosti, je základem pro analýzu omezených sítí a jejich jednotlivých vlastností chování.

Pokrývání Při generování grafu dosažitelnosti předem nevíme, jaké bude mít zkoumaná síť vlastnosti. Z toho důvodu obvykle nelze předem ani říct, zda se jedná o omezenou či neomezenou síť. Pokud by se jednalo o neomezenou síť, byla by množina dosažitelnosti nekonečná, a proto by byl nekonečný i graf dosažitelnosti. V takovém případě není možné použít graf dosažitelnosti k analýze Petriho sítě. I v dnešní době stále probíhá výzkum v oblasti neomezených Petriho sítí, v rámci kterého vznikají nové metody a postupy, jak vytvořit graf dosažitelnosti či jak jej analyzovat. Základem mnoha dalších prací se stalo zavedení symbolu ω Karpem a Millerem v [7], který reprezentuje nekonečnou množinu možných značení v konkrétním prvku vektoru značení.

Definice 18 *Značení $M = (m_1, m_2, \ldots, m_{|P|})$ pokrývá značení $N = (n_1, n_2, \ldots, n_{|P|})$, pokud platí $(\forall i)[m_i \geq n_i] \wedge (\exists i)[m_i > n_i]$.*

Definice 19 *Pokud počet tokenů v místě p_i může nabývat libovolně vysokých hodnot, potom místo m_i klademe ω. Pro počítání s ω platí:*

- $(\forall k)[\omega + k = \omega = \omega - k]$,
- $(\forall k)[\omega > k]$,

kde k je celé nezáporné číslo.

Pokud při generování stromu dosažitelnosti nastane situace, že nově nalezené značení pokrývá některé předchozí značení, stává se z grafu dosažitelnosti graf pokrytí, který je zobecněním grafu dosažitelnosti. Algoritmus generování grafu pokrytí se kvůli přítomnosti značení ω liší. Podrobněji se mu budeme věnovat v kapitole 3.3 věnované analýze neomezené sítě. Pokud při generování grafu dosažitelnosti není objeveno žádné pokrývající značení, je daná síť omezená, může být vygenerován její kompletní konečný graf dosažitelnosti a lze jej využít pro analýzu této Petriho sítě. Analýza na základě grafu dosažitelnosti je analýzou tohoto grafu, a proto se za účelem analýzy využívají grafové techniky, pomocí kterých lze určit následující vlastnosti.

3.2.2 Určení omezenosti

Omezenost je definovaná v definici 8. Jestli je síť omezená nebo neomezená, lze určit podle grafu dosažitelnosti, respektive grafu pokrytí. Pokud je graf dosažitelnosti konečný a neobsahuje žádný symbol ω, jedná se o omezenou síť. V takovém případě je možné analýzou grafu dosažitelnosti zjistit přesnou hodnotu omezenosti Petriho sítě a v důsledku také její bezpečnost. Algoritmus určení omezenosti je zřejmý. Je nutné projít celý graf dosažitelnosti a najít nejvyšší hodnotu počtu tokenů, která se v grafu vyskytuje. Za tímto účelem lze použít například *prohledávání grafu do šířky*. V praxi však úplně postačí projít seznam uzlů, pokud jej máme k dispozici,

protože při analýze omezenosti můžeme zcela ignorovat hrany i vzájemné propojení jednotlivých uzlů hranami.

Podle definice bezpečnosti, pokud je síť *1-omezená*, pak je síť bezpečná. Proto lze bezpečnost snadno určit na základě výsledku analýzy omezenosti.

3.2.3 Určení živosti

Pokud budeme na analýzu živosti Petriho sítě nahlížet jako na analýzu grafu, pak můžeme říct, že Petriho síť je živá, jestliže pro každou finální silně souvislou komponentu jejího grafu dosažitelnosti platí, že každý přechod této sítě je ohodnocením alespoň jedné hrany příslušné komponenty. To znamená, že každý přechod je znovu proveditelný. Silně souvislá komponenta je takový podgraf grafu, ve kterém mezi každými jeho dvěma vrcholy existuje cesta. Pokud z žádného vrcholu, který patří do silně souvislé komponenty, nevede žádná hrana do vrcholu, který do ní nepatří, potom se tato silně souvislá komponenta nazývá finální.

Definice 20 *Silně souvislá komponenta je podgraf $G'(V', E')$ orientovaného grafu G(V,E), pro jehož množinu vrcholů V' platí, že mezi každými dvěma uzly $u, v \in V'$ existuje cesta.*

Finální silně souvislá komponenta $G'(V', E')$ je taková, pro kterou navíc platí, že $(\forall v \in V')(\nexists w \in V \setminus V')[C(v, w)]$ kde C(x,y) je cesta mezi vrcholy x, y.

V [16] se jako algoritmus pro nalezení silně souvislých komponent uvádí algoritmus *prohledávání grafu do hloubky.*

Prohledávání grafu do hloubky Prohledávání grafu do hloubky (anglicky depth-first search) je druhý ze základních grafových algoritmů. Podle [16] tak spolu s algoritmem *prohledávání do šířky* tvoří základní dvojici algoritmů pro průchod grafem. Jsou proto základem mnoha dalších algoritmů pro práci s grafy. Poprvé algoritmus prohledávání do hloubky podle [2] použil Ch. P. Trémaux již v 19. století pro hledání cest z bludišť. Rozdíl mezi algoritmy prohledávání do šířky a do hloubky je v pořadí, v jakém navštíví jednotlivé vrcholy grafu. Podrobný popis je například

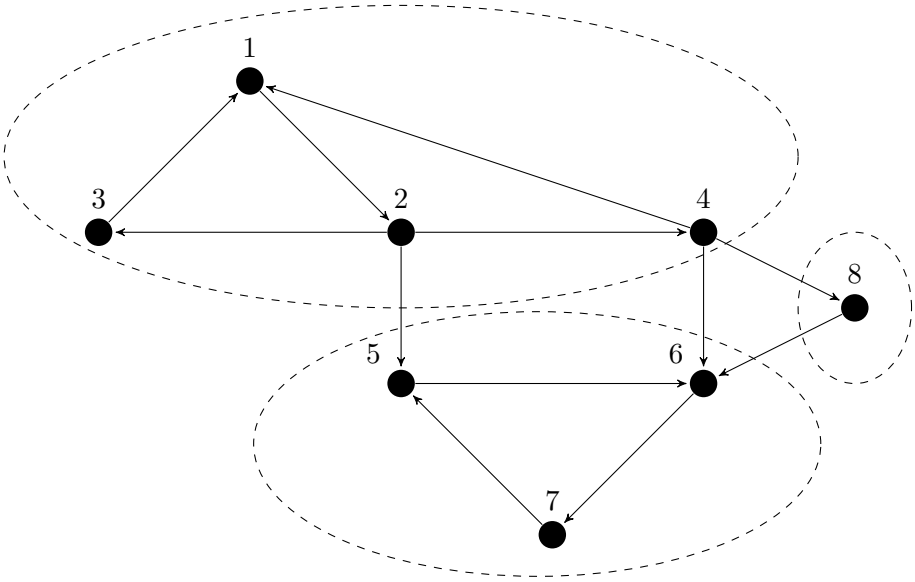
v [16] a algoritmus prohledávání do hloubky pro graf G a kořenový vrchol u vypadá následovně:

```
   Input: G, u
 1 Nastav všechny vrcholy grafu jako FRESH;
 2 Function DFS(G, u):
 3 |   Nastav u jako OPEN;
 4 |   Zpracuj vrchol u;
 5 |   for každý vrchol v, který je potomkem u, do
 6 |   |   Zpracuj hranu (u, v);
 7 |   |   if v je FRESH then
 8 |   |   |   Nastav u jako rodiče v;
 9 |   |   |   DFS(G,v);
10 |   |   end
11 |   end
12 |   Nastav u jako CLOSED;
```

Algoritmus 3: Prohledávání grafu do hloubky.

Nalezení silně souvislých komponent Jak již bylo řečeno, lze prohledávání do hloubky využít k nalezení silně souvislých komponent grafu. Prvním algoritmem, který tento problém řešil v lineárním čase, byl Tarjanův algoritmus, poprvé uvedený v [17]. Další popis Tarjanova algoritmu i s několika navrženými vylepšeními lze nalézt například v [14]. Tarjanův algoritmus využívá prohledávání do hloubky a přidává navíc indexování vrcholů podle pořadí, v jakém byly navštíveny. Při návratu z rekurze prohledávání do hloubky je pak každému vrcholu přiřazen nejnižší index vrcholu, do kterého z něj vede cesta. Vrchol s nejnižším indexem v komponentě se nazývá kořen této komponenty a všechny vrcholy se stejným indexem, jako je index kořene, patří do dané komponenty silné souvislosti. Výsledkem běhu Tarjanova algoritmu je tak množina silně souvislých komponent, přičemž počet prvků v této množině může být roven počtu vrcholů grafu. Velikost jednotlivých komponent může být různá a počet vrcholů v jedné silně souvislé komponentě může být roven 1, jak je vidět na obrázku 6. Konkrétní příklad běhu Tarjanova algoritmu je k vidění na obrázku 7.

Test živosti Po nalezení silně souvislých komponent v grafu dosažitelnosti je nutné určit finální silně souvislé komponenty. Zde se podle definice 20 nabízí více způsobů, jak ověřit, zda je silně souvislá komponenta i finální. Z pohledu Petriho sítě je taková komponenta množinou značení, mezi kterými lze neomezeně přecházet. V grafu dosažitelnosti jsou ohodnoceními hran přechody Petriho sítě, a proto, pokud je přechod součástí finální silně souvislé komponenty, je v ní vždy proveditelný, a tedy živý. Pokud pro všechny finální silně souvislé komponenty grafu platí, že alespoň jednou obsahují každý přechod, pak je síť živá. Protože tyto komponenty jsou podgrafy grafu dosažitelnosti, můžeme na ně aplikovat algoritmy prohledávání do šířky i do hloubky, přičemž budeme ověřovat, zda je každý přechod ohodnocením alespoň jedné hrany v



Obrázek 6: Ukázka silně souvislých komponent grafu (převzato z [16]).

každé komponentě. Podobně jako u analýzy omezenosti ovšem i zde postačí analyzovat pouze seznam hran uvnitř jednotlivých komponent, pokud takový seznam máme k dispozici.

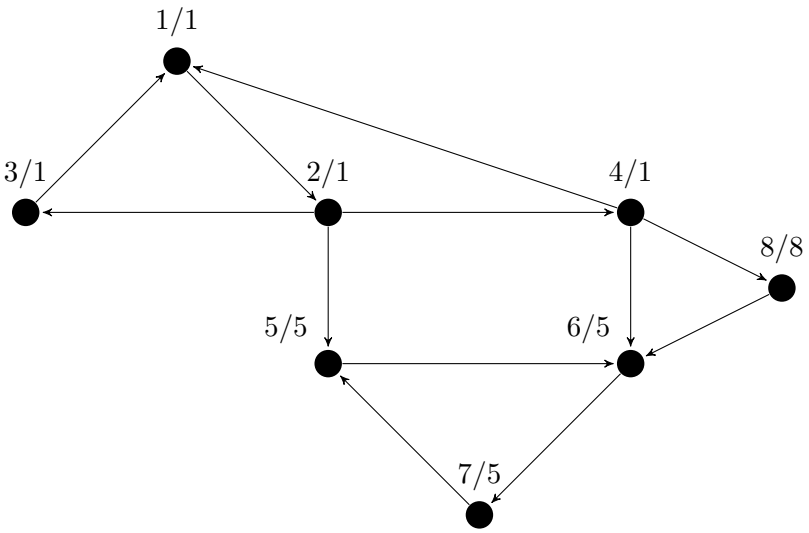
3.2.4 Určení reverzibility

Podle definice v kapitole 2.4.3 je Petriho síť reverzibilní, pokud je její počáteční značení dosažitelné z každého dosažitelného značení sítě. Při analýze na základě grafu dosažitelnosti to znamená, že z kořenového vrcholu reprezentujícího počáteční značení vede orientovaná cesta do všech dosažitelných značení. Aby byla síť reverzibilní, musí existovat také orientovaná cesta ze všech dosažitelných značení Petriho sítě zpět do počátečního, tedy z každého vrcholu grafu zpět do kořenového vrcholu. Pokud budeme na určení reverzibility nahlížet jako na grafový problém, pak platí, že aby byla Petriho síť reverzibilní, musí být její graf dosažitelnosti silně souvislý.

Pro analýzu je možné použít podobný postup jako pro analýzu živosti. Pomocí Tarjanova algoritmu nalezneme silně souvislé komponenty grafu dosažitelnosti, jak je popsáno v kapitole 12. Pokud výsledkem běhu Tarjanova algoritmu bude pouze jediná silně souvislá komponenta, pak je celý graf silně souvislý a Petriho síť je reverzibilní.

3.2.5 Existence uzamčení

Uzamčení je dosažitelné značení Petriho sítě, ve kterém není žádný přechod proveditelný. V kontextu grafu dosažitelnosti má takové značení podobu vrcholu, do kterého může vést libovolný počet vstupních hran, nemá ovšem žádnou výstupní hranu. Analýzu existence uzamčení je možné provést analýzou jejího grafu dosažitelnosti. Lze k tomu využít algoritmy prohledávání grafu do šířky i do hloubky, přičemž při zpracování každého vrcholu nás bude zajímat počet následníků. Vrchol bez následníků reprezentuje uzamčení, a pokud takový vrchol nalezneme, pak síť obsahuje



Obrázek 7: Průběh Tarjanova algoritmu. Čísla nalevo označují pořadí navštívení příslušného vrcholu algoritmem, čísla vpravo pak příslušnost k silně souvislé komponentě.

uzamčení. Jak je uvedeno v kapitole 2.4.4, existence uzamčení jednoznačně říká, že Petriho síť není živá ani reverzibilní.

3.2.6 Určení repetičnosti

Repetičnost Petriho sítě je popsaná v kapitole 2.4.5. Její definice říká, že musí existovat sekvence přechodů σ z počátečního značení M_0 taková, že obsahuje každý přechod sítě nekonečně mnohokrát. To znamená, že graf dosažitelnosti Petriho sítě musí obsahovat uzavřený sled, v němž se vyskytuje každý přechod alespoň jednou, jak je napsáno v [1]. Existence takového uzavřeného sledu znamená, že je možné provést příslušnou sekvenci přechodů a vrátit se zpět do konkrétního značení, ve kterém uzavřený sled začal. Takovouto sekvenci je pak logicky možné provádět donekonečna, což umožňuje splnění podmínky repetičnosti.

Pro určení repetičnosti na základě grafu dosažitelnosti je zapotřebí nalézt uzavřené sledy v grafu. Za tímto účelem jsou nejvhodnější algoritmy vycházející z *prohledávání grafu do hloubky*. Johnsonův algoritmus, uvedený v [5], je efektivní při detekci všech elementárních cyklů. Elementární cyklus je jiný název pro uzavřený sled. V literatuře je tak někdy možné narazit i na definici, že elementární cyklus obsahuje každý vrchol kromě počátečního právě jednou a počáteční vrchol obsahuje právě dvakrát. Pro analýzu repetičnosti ovšem není nutné detekovat elementární cykly, protože podle definice se může jednat o jakýkoli uzavřený sled. Podmínky repetičnosti nevyžadují, aby byl v uzavřeném sledu každý vrchol právě jedenkrát. Z tohoto důvodu je dostačující nalezení silně souvislých komponent grafu, protože každá taková komponenta, která obsahuje více než jeden vrchol, obsahuje cyklus a zároveň jsou všechny její vrcholy součástí uzavřeného sledu. Navíc platí, že uvnitř silně souvislé komponenty může být více než jeden cyklus. Jak je uvedeno v kapitole 3.2.3, k detekci silně souvislých komponent je vhodný Tarjanův algoritmus.

Pokud mezi takto nalezenými silně souvislými komponentami nalezneme alespoň jednu, která obsahuje každý přechod Petriho sítě alespoň jednou, pak je tato síť repetiční.

Drobným nedostatkem použití Tarjanova algoritmu pro test repetičnosti je fakt, že každý vrchol zařadí do silně souvislé komponenty, a proto vrcholy, které nejsou součástí žádného cyklu, skončí v jednoprvkových komponentách. V jednoprvkové silně souvislé komponentě však může skončit také vrchol, jehož jedna výstupní hrana je zároveň i jeho vstupní hranou. Jedná se o případ, kdy se provedením přechodu značení sítě nijak nezmění a vrchol je tak ve smyčce sám se sebou. Tento případ je však relevantní pouze v případě elementární sítě, kdy by existence takovéto smyčky znamenala důkaz repetičnosti. V sítích s více než jedním přechodem je jednoprvková silně souvislá komponenta pro analýzu repetičnosti nepodstatná.

3.2.7 Určení konzistenčnosti

Jelikož se jedná o speciální případ repetičnosti, metoda analýzy je podobná. Rozdíl je v tom, že konzistenčnost vyžaduje existenci sekvence přechodů, vedoucí z M_0 zpět do M_0, takovou, že obsahuje každý přechod sítě alespoň jednou. Pokud tedy silně souvislá komponenta, nalezená Tarjanovým algoritmem, ve které je vrchol reprezentující značení M_0, obsahuje každý přechod sítě alespoň jednou, pak je Petriho síť konzistentní.

3.2.8 Test dosažitelnosti

Dosažitelnost značení je typický problém analýzy Petriho sítí. Jedná se o otázku, zda se síť může z počátečního značení dostat do nějakého konkrétního značení, tedy jestli se modelovaný systém může dostat do konkrétního stavu. Pro omezené sítě je tento problém relativně snadno řešitelný, ovšem v neomezených sítích se může jednat o poměrně složitý problém. V této kapitole se však zabýváme pouze analýzou sítí omezených. Jelikož omezená Petriho síť má konečnou množinu dosažitelnosti a konečný graf dosažitelnosti, lze dosažitelnost značení otestovat pomocí některého algoritmu pro průchod grafem, a to jak prohledávání do šířky, tak prohledávání do hloubky. V obou případech nás v každém vrcholu bude zajímat, zda značení reprezentované tímto vrcholem odpovídá hledanému značení.

3.2.9 Určení konzervativnosti

V původní práci je konzervativnost testována na základě *P-invariantů*.

Definice 21 *P-invariantem struktury* $\langle P, T, I, O\rangle$ *se nazývá takový vektor* $Y = (y_1, y_2, \ldots, y_{|P|})^T$, *kde* $y_i \in N_0$, *který zleva nuluje matici incidence C. Platí pro něj* $Y^T.C = 0^T$.
Množina $P_Y = \{p_i \in P : y_i > 0\}$ *se nazývá nosič P-invariantu Y.*

Pokud existuje P-invariant, jehož nosič se shoduje s množinou míst Petriho sítě, pak je tato síť konzervativní.

Dalším způsobem analýzy je využití stromu dosažitelnosti pro výpočet váhy tokenů. Pokud existuje váhový vektor $w(n \times 1)$, pro který platí $w^T m_0 = w^T m_i$ pro $i = \{1, 2, \ldots, |RG| - 1\}$, potom je síť konzervativní. Tato metoda však vede k problému lineárního programování, kde je potřeba sestavit rovnici pro každý vrchol stromu dosažitelnosti za účelem nalezení váhového vektoru w. Protože počet vrcholů stromu dosažitelnosti může nabývat velmi vysokých hodnot, nejeví se tento postup jako příliš efektivní.

3.3 Analýza neomezené sítě

Pro neomezenou síť platí, že alespoň jedno její místo je neomezené. Počet tokenů v takovém místě může nabývat libovolně vysokých hodnot, což má za následek nekonečnost množiny dosažitelnosti. Z toho dále vyplývá, že také graf dosažitelnosti bude nekonečný. Využití grafu dosažitelnosti k analýze neomezené Petriho sítě je tak samozřejmě nemožné a musí být použit jiný přístup. Analýza neomezených Petriho sítí se tak v konečném důsledku stává velmi komplikovaným problémem, na jehož výzkumu se dodnes pracuje a stále vznikají nové postupy analýzy. Jak je uvedeno již v kapitole 3.2.1 na straně 23, základy analýzy neomezených sítí položili v [7] Karp s Millerem, kteří zavedli nový symbol ω pro vyznačení pokrývajícího značení. Příslušné definice pokrývání a počítání s ω symbolem jsou uvedeny v definicích 18 a 19. Zavedení ω symbolu do značení Petriho sítí umožňuje vytvoření konečné varianty grafu dosažitelnosti, která se nazývá graf pokrytí.

Konstrukce stromu pokrytí Algoritmus konstrukce stromu pokrytí vypadá podle [13] následovně:

Input: PN, M_0

Output: Graf pokrytí

1 Počáteční značení M_0 označ jako *NEW* a nastav jako kořen stromu;
2 **while** *existuje značení M označené jako* NEW **do**
3 Vyber značení M s příznakem *NEW*;
4 Pokud je M totožné se značením, které se již vyskytlo na cestě z M_0 do M, označ M jako *OLD* a pokračuj s novým značením;
5 Pokud $E(M) = \emptyset$, tedy pokud ve značení M není proveditelný žádný přechod, je vrchol M terminálním vrcholem stromu;
6 **if** $E(M) \neq \emptyset$, *pak pro každý proveditelný přechod* t **then**
7 Získej značení M' vzniklé provedením přechodu t ve značení M;
8 Pokud na cestě z M_0 do M existuje značení M'' takové, že $M'(p) \geq M''(p)$ pro každé místo p a $M' \neq M''$, potom značení M' pokrývá značení M''. Pro každé místo p takové, že $M'(p) > M''(p)$, nahraď $M'(p)$ symbolem ω.;
9 Přidej do stromu nový vrchol M', hranu z M do M' s ohodnocením t a označ M' jako *NEW*;
10 **end**
11 **end**

Algoritmus 4: Konstrukce stromu pokrytí.

Takto vzniklý strom pokrytí lze pak snadno převést na graf pokrytí v případě potřeby, případně lze upravit samotný algoritmus, aby generoval graf pokrytí. Jak je vidět z tohoto algoritmu, graf pokrytí vzniká velmi podobně jako graf dosažitelnosti a také vychází z *prohledávání grafu do šířky*, kdy ve vlnách nachází nová dosažitelná značení. Je tedy možné použít jeden algoritmus, jehož výsledkem bude buď graf dosažitelnosti nebo graf pokrytí v závislosti na omezenosti Petriho sítě.

Problémem grafu pokrytí pro potřeby analýzy je právě symbol ω. Umožňuje sice vytvořit konečnou reprezentaci nekonečné množiny dosažitelnosti, ovšem za cenu ztráty informací. Nelze určit, jaké hodnoty se pod symbolem ω mohou skrývat. Může reprezentovat například pouze liché nebo pouze sudé hodnoty. Ve [13] je uveden příklad dvou Petriho sítí, které jsou odlišné, mají různé vlastnosti, ale přesto mají stejný graf pokrytí. Proto má graf pokrytí jen velmi omezené využití při analýze Petriho sítí. Podle [13] lze graf pokrytí použít pro detekci těchto vlastností:

- Omezenost,
- Bezpečnost,
- Existence mrtvého přechodu (není nikdy proveditelný).

Strom pokrytí, označovaný také zkratkou FRT (anglicky Finite Reachability Tree), ovšem kvůli ztrátě informací nelze využít pro analýzu vlastností jako jsou živost, reverzibilita, dosažitelnost značení nebo existence uzamčení. V důsledku tohoto velmi omezeného využití tak vznikly nové varianty FRT. V rámci snahy o rozšíření možností využití pro analýzu živosti vznikl v [3] a [4] Augmented Reachability Tree (ART). ART je použitelný pro analýzu třídy Petriho sítí, která se nazývá *one-place-unbounded*. Petriho sítě spadající do této třídy mají maximálně jedno neomezené místo, což značně limituje rozsah použitelnosti ART, přestože je tato varianta užitečná při analýze živosti.

Jiné modifikace jsou zaměřeny více obecně. V [19] a [20] byl představen Modified Reachability Tree (MRT), ve kterém je místo pouhého symbolu ω použit výraz $k\omega_n + q$, který tak v sobě uchovává více informací. MRT lze využít pro detekci uzamčení, řešení problému dosažitelnosti značení a v omezené míře také k analýze živosti. Později bylo v [15] ukázáno, že test existence uzamčení, založený na MRT, obsahuje chybu. V návaznosti na to byl v [21] představen New Modified Reachability Tree (NMRT), který odstraňuje nedostatky MRT a umožňuje detekci uzamčení a řešení dosažitelnosti značení pro poměrně rozsáhlou množinu Petriho sítí. Výzkum však pokračuje i nadále a v [23] byla uvedena metoda pro analýzu živosti založená na NMRT. Zapracování NMRT a využití jeho možností analýzy v rozšiřovaném programu budeme podrobněji popisovat v kapitole 4.

4 Generování stromu dosažitelnosti pro neomezené sítě

Jak bylo řečeno v předchozích částech práce, analýza omezených sítí je dobře řešitelná pomocí již existujících metod. Pomocí grafu dosažitelnosti nebo incidenční matice lze příslušnými algoritmy otestovat vlastnosti omezených Petriho sítí. Největším problémem je případná časová nebo paměťová náročnost, což závisí na dané Petriho síti. Oblast neomezených Petriho sítí je však stále předmětem aktivního výzkumu a i nadále zůstává otevřena řada otázek a problémů. Možnosti analýzy neomezených Petriho sítí jsou stále velmi limitované, přestože došlo k jistým pokrokům. Jedním z těchto pokroků je zajisté varianta stromu dosažitelnosti zvaná *NMRT*, která je ve srovnání s obdobnými metodami použitelná na větší množinu Petriho sítí, nikoliv pouze na *one-place-unbounded* Petriho sítě. Aplikace NMRT a dalších navazujících algoritmů do existujícího programu je dalším bodem této práce.

4.1 Generování NMRT

Ještě před uvedením samotného algoritmu pro generování NMRT je potřeba zavést několik pojmů uvedených v [21]. Základem NMRT jsou $\omega-$*čísla*, která rozšiřují možnosti již existujícího symbolu ω a umožňují uchovat více informací o stavu tokenů v příslušném místě.

Definice 22 *Množina celých čísel S se nazývá $\omega-$*číslo*, pokud $\forall k \in \mathbb{N}^{+}; n, q \in \mathbb{Z} : S = \{ik+q | i \geq n, 0 \leq q < k\}, S \in \mathbb{Z}$.*
S lze vyjádřit jako $S = \omega(k, n, q) \equiv k\omega_n + q$, kde k je základ, n je nejnižší mez a q je zbytek.

Například $\omega-$*číslo* $\omega(1,0,0)$ reprezentuje množinu celých čísel $\omega_0 = \{0,1,2,3,\ldots\}$, $\omega(2,1,0) = 2\omega_1 = \{2,4,6,8,\ldots\}$ a $\omega(3,2,1) = 3\omega_2 + 1 = \{7,10,13,16,\ldots\}$. Jak je vidět z příkladů, $\omega-$*číslo* je množina celých čísel, která reprezentuje množinu možných hodnot tokenů v konkrétním místě Petriho sítě. Lze tak zachytit informaci o možných značeních.

Pro dvě $\omega-$*čísla* $a = \omega(k,m,q)$ a $b = \omega(k,n,q)$ kde $m \leq n$ platí, že $a \leq b$. Pokud bychom brali a a b jako množiny, potom platí $b \subseteq a$.

Definice 23 *Vektor se nazývá $\omega-$vektor, pokud je alespoň jeden jeho prvek $\omega-$*číslo*.*

Příklad $\omega-$*vektoru*: $(1, 3, \omega_1, 0)$.

Definice 24 *Pokud je značení M reprezentováno $\omega-$vektorem, potom se M nazývá $\omega-$značení.*

Každé $\omega-$*značení* reprezentuje množinu běžných značení. Mějme například dvě značení $M_1 = (1, 0, 2\omega_1)$ a $M_2 = (1, 0, 2\omega_2)$. Protože $2\omega_1 = \{2,4,6,8,\ldots\}$ a $2\omega_2 = \{4,6,8,10,\ldots\}$, platí, že $M_1 = (1,0,2\omega_1) = \{(1,0,2),(1,0,4),(1,0,6),\ldots\}$ a $M_2 = (1,0,2\omega_2) = \{(1,0,4),(1,0,6),\ldots\}$. Pro tato dvě značení platí, že $M_1 \leq M_2$ a že $M_2 \subseteq M_1$.

Skutečnost, že jediné značení může představovat nekonečnou množinu běžných značení, způsobuje problém při určování proveditelnosti přechodu, který může být proveditelný pouze pro určitou podmnožinu běžných značení reprezentovaných $\omega-$*značením*. Důsledkem je zavedení *podmíněné proveditelnosti přechodu.*

Definice 25 *Přechod t je podmíněně proveditelný v $\omega-$značení M, pokud není proveditelný v alespoň jednom běžném značení z M a zároveň je proveditelný ve všech ostatních značeních z M. Aby byl přechod podmíněně proveditelný, musí být proveditelný alespoň v jednom z běžných značení z M.*

Definice 26 *Definice proveditelnosti přechodu je upravena vzhledem k existenci $\omega-$značení tak, že přechod je proveditelný v $\omega-$značení M, pokud je proveditelný ve všech běžných značeních z M. Obdobně přechod není proveditelný, pokud není proveditelný v žádném běžném značení z M.*

Existence $\omega-$čí*sel* také komplikuje výpočet nového značení po provedení přechodu. Dochází totiž k přičítání hodnot k nekonečné množině celých čísel. Proto je v práci [21] zavedena *funkce potomka.*

4.1.1 Funkce potomka

Next-state function neboli *funkce potomka* slouží pro výpočet značení vzniklého provedením přechodu. Značí se $\delta(M,t)$, kde M je značení Petriho sítě a t je přechod proveditelný či podmíněně proveditelný v M. Funkce potomka $\delta(M,t)$ představuje značení vzniklé provedením přechodu t ve značení M. Pro samotný výpočet funkce potomka je nezbytné definovat sčítání $\omega-$čí*sel* a celých čísel, aby bylo možno korektně určit novou hodnotu značení po provedení přechodu.

Definice 27 *Sčítání ω-čísel a celých čísel je definováno následovně:*
Mějme ω-číslo $\omega(k,n,q)$ a číslo $a \in Z$. Potom platí $\omega(k,n,q)+a=\omega(k,n+s,r)$, kde
$q+a=s.k+r, s \in Z, 0 \leq r < k$.

Vezmeme-li například ω-číslo $\omega(2,0,1)=\{1,3,5,\ldots\}$ a číslo $a=2$, pak podle definice sčítání jejich součet vypadá takto:
$\omega(2,0,1)+a=\omega(2,1,1)=\{3,5,7,9\ldots\}$.

Výpočet nového značení funkcí potomka lze rozdělit na tři varianty. Mějme značení M a přechod $t \in T$.

1. Pokud je M běžné značení a t je proveditelný v M, potom je $\delta(M,t)$ vypočítána pomocí pravidel provedení přechodu v běžném značení.

2. Pokud je M $\omega-$*značení* a t je proveditelný v M, potom je $\delta(M,t)$ vypočítána pomocí pravidel provedení přechodu v běžném značení a pomocí sčítání $\omega-$čí*sel* a celých čísel.

Na celočíselné prvky příslušného $\omega - vektoru$ je tedy použito sčítání celých čísel, zatímco na $\omega - čísla$ je aplikováno sčítání $\omega - čísel$ a celých čísel.

3. Pokud je M $\omega - značení$ a t je podmíněně proveditelný v M, potom je $\delta(M, t)$ vypočítána následovně:

   - Ze značení M jsou odstraněna všechna běžná značení, ve kterých je t neproveditelný, čímž vznikne značení M',
   - Podle kroku 2. je vypočítána $\delta(M', t)$ a poté platí, že $\delta(M, t) = \delta(M', t)$.

Pokud je tedy přechod pouze podmíněně proveditelný, je dané $\omega - značení$ upraveno tak, aby v něm byl přechod proveditelný, a následně je vypočítáno nové značení podle odpovídajících pravidel.

4.1.2 Algoritmus pro generování NMRT

V práci [21] je uveden algoritmus, pomocí kterého je možno vygenerovat NMRT. Takto vzniklý NMRT měl ovšem jednu velkou nevýhodu v počtu svých vrcholů, jelikož obsahoval velké množství duplicitních, a tedy často zbytečných, vrcholů. Důsledkem byla zbytečně větší časová i paměťová náročnost generování NMRT. Z těchto důvodů byl v práci [22] uveden modifikovaný algoritmus pro vytvoření redukovaného NMRT. Výsledkem tohoto algoritmu je výrazné snížení počtu duplicitních vrcholů NMRT, aniž by došlo k omezení aplikovatelnosti či možností analýzy. Proto bude v této práci uveden algoritmus pro redukovaný NMRT z [22].

Graf NMRT obsahuje kromě normálního ještě tři další typy vrcholů - terminální, duplicitní a ω-duplicitní. Terminální vrchol reprezentuje uzamčení, tedy značení, ve kterém není žádný přechod proveditelný. Duplicitní vrchol je takový, jehož značení již obsahoval algoritmem dříve zpracovaný vrchol. Vrchol, reprezentující $\omega - značení$, které je zcela obsaženo v $\omega - značení$ již dříve zpracovaného vrcholu, se nazývá ω-duplicitní.

Input: PN, M_0
Output: NMRT

Nechť je x_0 kořenový vrchol NMRT a M_0 značením vrcholu x_0;
Inicializuj *stack* X = $\{x_0\}$ a *set* S = $\{M_0\}$;
while *X je neprázdný* **do**
    x = *pop(X)*;
    Nechť M_x je značením vrcholu x;
    foreach $t \in T$ **do**
        if *t je proveditelný nebo podmíněně proveditelný v* M_x **then**
            Vypočítej funkci potomka $\delta(M_x, t)$ a vytvoř nový vrchol z;
            if *existuje vrchol y na cestě z* x_0 *do x takový, že* $\delta(M_x, t) > M_y$ **then**
                foreach $p \in P$ **do**
                    if $\delta(M_x, t)_p > (M_y)_p$ *a* $\delta(M_x, t)_p \in N$ **then**
                        $(M_z)_p = \omega(k, n, q)$, kde $k = \delta(M_x, t)_p - (M_y)_p$, $\delta(M_x, t)_p = nk + q$ a $0 \leq q < k$;
                    else
                        $(M_z)_p = \delta(M_x, t)_p$
                    end
                end
            else
                $M_z = \delta(M_x, t)$
            end
            $S = S \cup M_z$;
            if *z je normální vrchol* **then**
                X = *push(X,z)*
            end
            if *t je proveditelný v* M_x **then**
                Přidej hranu t z vrcholu x do vrcholu z;
            else
                Přidej přerušovanou hranu t z vrcholu x do vrcholu z;
            end
        end
    end
end

Algoritmus 5: Konstrukce NMRT.

4.1.3 Omezení NMRT

Přestože $\omega - \check{c}\acute{i}sla$ umožňují uchování více informací než pouze symbol ω, stále nezachycují dostatek informací, aby mohl být NMRT použit na jakoukoliv Petriho síť bez rizika chybných výsledků. V práci [21] jsou definovány dvě třídy Petriho sítí - $\omega - z\acute{a}visl\acute{e}$ a $\omega - nez\acute{a}visl\acute{e}$, přičemž NMRT poskytuje korektní výsledky pouze pro analýzu $\omega - nez\acute{a}visl\acute{y}ch$ Petriho sítí.

Definice 28 *Mějme vrcholy x a z v NMRT, reprezentující značení M_x a M_z, takové, že M_z je značení vzniklé provedením přechodu t ve značení M_x a zároveň $M_z \neq \delta(M_x, t)$.*

M_z se nazývá $\omega - z\acute{a}visl\acute{e}$, pokud existuje vrchol y na cestě z kořenového vrcholu do z takový, že $\delta(M_x, t) > M_y$ a zároveň alespoň dva prvky v $\delta(M_x, t)$ jsou větší než odpovídající prvky v M_y.

Petriho síť se nazývá $\omega - z\acute{a}visl\acute{a}$, pokud obsahuje alespoň jedno $\omega - z\acute{a}visl\acute{e}$ značení. Pokud Petriho síť neobsahuje žádné $\omega - z\acute{a}visl\acute{e}$ značení, nazývá se $\omega - nez\acute{a}visl\acute{a}$.

Definice 28 zjednodušeně říká, že pokud nějaké značení M_i, při jehož výpočtu došlo ke transformaci alespoň jednoho celého čísla na $\omega - \check{c}\acute{i}slo$, pokrývá jiné značení M_j a má více tokenů alespoň ve dvou místech, potom se jedná o $\omega - z\acute{a}vislou$ síť. V takovém případě nelze pomocí $\omega - \check{c}\acute{i}sel$ zachytit vzájemný vztah mezi těmito dvěma místy a jejich možnými počty tokenů.

Jednoduchý příklad $\omega - z\acute{a}visl\acute{e}$ Petriho sítě je na obrázku 8. Z počátečního značení (0,0) se síť provedením přechodu *t2* dostane do značení (1,1). Toto značení evidentně pokrývá značení (0,0), takže se použitím $\omega - \check{c}\acute{i}sel$ změní na (ω_1, ω_1). Při pohledu na Petriho síť na obrázku 8 je zřejmé, že v místě *p2* nikdy nemůže být více tokenů, než v místě *p1*. Podle definice $\omega - \check{c}\acute{i}sel$ však značení (ω_1, ω_1) reprezentuje například i značení (10,20), které je ovšem ve skutečnosti nedosažitelné. V $\omega - \check{c}\acute{i}slech$ je totiž ztracena informace, že při provedení přechodu *t2* se zvýší počet tokenů v místech *p1* i *p2* o 1.

Obrázek 8: Příklad $\omega - z\acute{a}visl\acute{e}$ Petriho sítě.

4.2 Aplikace NMRT

Podle [21] a [22] lze NMRT použít pro určení dosažitelnosti značení a hledání uzamčení, což lze aplikovat při ověřování živosti Petriho sítě. V [21] jsou uvedeny příslušné důkazy konečnosti NMRT, důkaz korektnosti hledání uzamčení a také důkaz, že NMRT obsahuje všechna dosažitelná a žádná nedosažitelná značení.

4.2.1 Dosažitelnost značení

Díky skutečnosti, že NMRT obsahuje všechna dosažitelná značení, je možné určit dosažitelnost konkrétního značení i pro neomezené Petriho sítě. Běžný strom pokrytí pro test dosažitelnosti použít nelze, protože symbol ω neobsahuje informaci o rychlosti přibývání tokenů v daném místě. Nelze tedy určit, jaké hodnoty tokenů mohou v daném místě být. Naproti tomu $\omega - \check{c}\acute{i}sla$ přesně popisují množinu možných hodnot tokenů v příslušném místě, což umožňuje provedení testu dosažitelnosti.

4.2.2 Nalezení uzamčení

Na základě NMRT lze určit, zda v analyzované Petriho síti existuje uzamčení. Uzamčení může mít v NMRT dvě podoby, jak se uvádí v [21]. První variantou je *terminální* značení, ve kterém není žádný přechod proveditelný. Druhou variantou je existence *zcela podmíněného značení*. Pro takové značení platí, že v něm není žádný přechod proveditelný a zároveň je alespoň jeden přechod podmíněně proveditelný. Pokud totiž existuje $\omega - zna\check{c}en\acute{i}$, ve kterém je podmíněně proveditelný přechod, pak platí, že v některých běžných značeních je tento přechod neproveditelný. Pokud v daném $\omega - zna\check{c}en\acute{i}$ není žádný proveditelný přechod, může nastat situace v určitém dosažitelném běžném značení taková, že se bude jednat o uzamčení.

4.2.3 Využití NMRT v testech vlastností

NMRT lze částečně využít pro test živosti, reverzibility či existence uzamčení. Není sice možné prokázat, že je daná síť živá či reverzibilní, ovšem díky detekci uzamčení je možné určit, že síť živá, respektive reverzibilní, není.

Existence uzamčení je jednou z vlastností Petriho sítí, kterou je možné testovat již v původním programu. Pro test existence uzamčení lze využít NMRT, čímž lze získat korektní výsledky i pro třídu $\omega - nez\acute{a}vislých$ Petriho sítí.

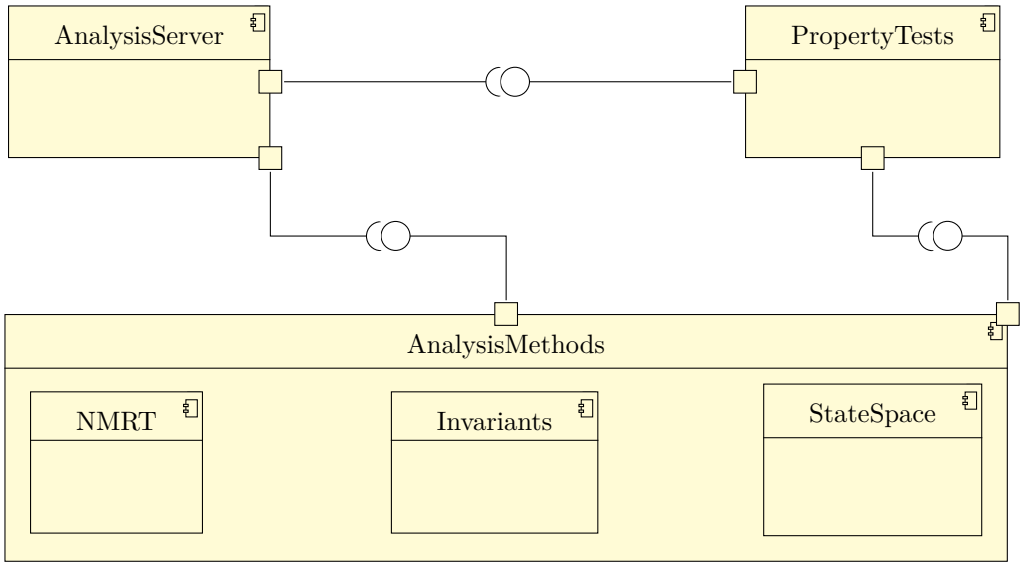
Pro živost platí, že pokud v Petriho síti existuje uzamčení, potom tato síť není živá, protože nesplňuje definici 7. Pokud existuje značení v NMRT, které je uzamčením, lze s jistotou říci, že daná Petriho síť není živá.

Existuje - li v Petriho síti značení M, které je uzamčením, platí, že počáteční značení M_0 je nedosažitelné ze značení M. V takovém případě Petriho síť nesplňuje definici 9. Pokud existuje značení v NMRT, které je uzamčením, lze také s jistotou říci, že analyzovaná Petriho síť není reverzibilní.

4.2.4 Implementace NMRT

Generování NMRT je implementováno jako další analytická metoda, která je přidána do již existující množiny metod analytického serveru. Přidání metody do množiny existujících metod zajišťuje registrace pomocí klíčového slova a příslušné konstruktorové funkce. Třídy potřebné

pro vygenerování NMRT se nacházejí v balíku *cz.diploma.analysis.methods.nmrt*. Samotné generování NMRT je realizováno třídou *NMRTConstructor*, která obsahuje potřebné metody a ve které je naimplementován algoritmus 5. Začlenění NMRT do stávajícího programu je znázorněno na obrázku 9. Jednotlivé analytické metody jsou na sobě navzájem nezávislé, protože uživatel má možnost si pro účely analýzy zvolit jejich libovolnou podmnožinu.



Obrázek 9: Ukázka začlenění NMRT mezi analytické metody serveru.

Při analýze Petriho sítí pomocí stromu dosažitelnosti či NMRT může být velkým problémem paměťová či časová náročnost díky *explozi stavového prostoru*. I zdánlivě jednoduchá Petriho síť může mít v závislosti na počátečním značení obrovský počet dosažitelných značení. Při implementaci byl proto kladen důraz na minimalizaci paměťových a časových nároků. Zvolené řešení využívá převážně primitivní datové typy a speciální implementace kolekcí pracujících právě s primitivními datovými typy. Tyto kolekce jsou součástí knihovny *Trove*.

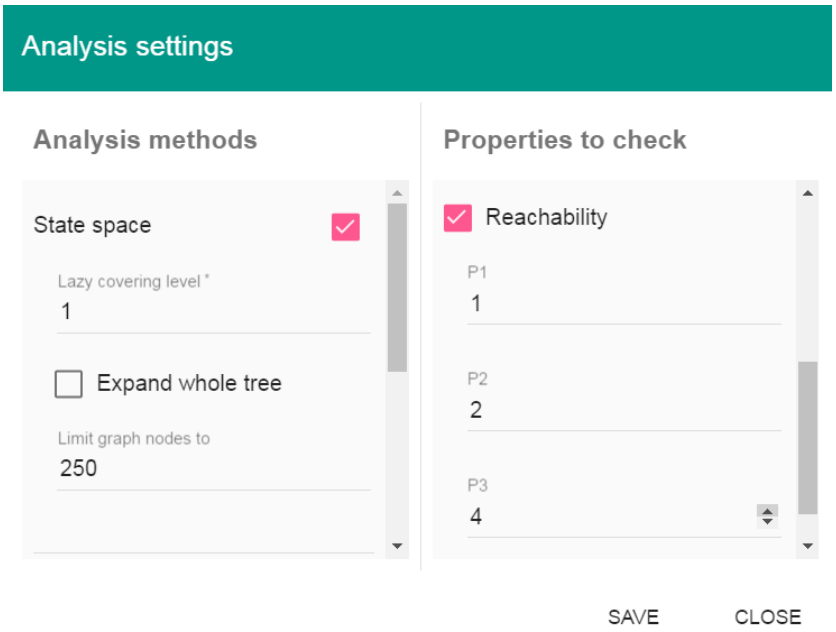
Vygenerovaný NMRT je přidán k výsledkům analytických metod a je pak dále využíván při ověřování vlastností Petriho sítě. Vzhledem k možnostem uplatnění NMRT jsme přidali testy živosti, reverzibility a existence uzamčení do již existujících tříd zajišťujících testování daných vlastností. Tyto testy jsou realizovány metodami ověřujícími existenci uzamčení ve vygenerovaném NMRT. Vzhledem k důkazům uvedeným v [21] považujeme ověřování existence uzamčení pomocí NMRT za metodu nadřazenou metodám v původní verzi nástroje. *Líný strom pokrytí* je sice univerzálně použitelný, ovšem jeho výsledky jsou závislé na použitém nastavení úrovně pokrytí. Při nedostatečné úrovni může dojít k chybným výsledkům analýzy. V důsledku je proto použit výsledek testu na základě NMRT k případné změně celkového výsledku, protože jeho

správnost není závislá na použitém nastavení. Pokud je tedy možné rozhodnout o konkrétní vlastnosti pomocí NMRT, pak je za celkový výsledek testu této vlastnosti považován výsledek na základě NMRT. To platí i v případě, že z jiných testů, například na základě stromu dosažitelnosti, získáme výsledek opačný. Dalším podkladem pro tento způsob aplikace jsou příklady, kdy doposud existující metody chybně určí vlastnosti Petriho sítě, zatímco přidáním NMRT mezi použité metody dojde k určení správného výsledku. Tyto příklady uvádíme v kapitole 6.

Výsledky jednotlivých testů jsou vždy zobrazeny uživateli v editoru včetně konkrétních důvodů. V důsledku našeho zapracování NMRT a způsobu jeho aplikace může dojít k vypsání navzájem se vylučujících důvodů pro výsledný status testu. Toto se týká výsledků na základě NMRT a na základě *líného grafu pokrytí*. Proto jsme k vypisovaným důvodům u analýzy na základě *líného grafu pokrytí* přidali informaci o použité úrovni pokrytí. Pokud je totiž tento výsledek chybný, je to z důvodu nedostatečné úrovně pokrytí.

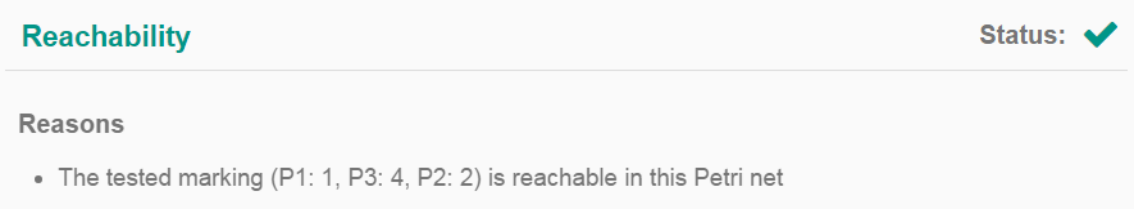
4.2.5 Test dosažitelnosti

Dalším využitím NMRT je možnost testu dosažitelnosti zvoleného značení, která v původním programu nebyla implementována. Proto jsme se rozhodli pro její doplnění do množiny testovatelných vlastností Petriho sítí. Uživatel má možnost volby testovaného značení v editoru v nastavení analýzy. Na obrázku 10 je ukázka příslušné volby. Jak je vidět na obrázku 11, výsledky testu dosažitelnosti jsou zobrazeny mezi ostatními testovanými vlastnostmi.



Obrázek 10: Ukázka volby značení pro test dosažitelnosti (screenshot).

Výsledek obsahuje status testu, zda je testované značení dosažitelné či nikoliv, a také příslušné odůvodnění, jehož součástí je i testované značení.



Obrázek 11: Ukázka výsledků testu dosažitelnosti (screenshot).

Na výpočetním serveru je test dosažitelnosti implementován jako další testovaná vlastnost. Příslušná konstruktorová funkce je zaregistrována standardním, způsobem stejně jako je tomu u ostatních vlastností. Během testování dosažitelnosti je z parametrů analýzy získáno požadované značení. Pro ověření dosažitelnosti jsme implementovali jedinou metodu, která rozhodne na základě NMRT. Pokud NMRT není k dispozici nebo pokud jej není možné použít bez rizika chybných výsledků, nelze o dosažitelnosti značení rozhodnout. Výsledek včetně odůvodnění je přidán k výsledkům ostatních testovaných vlastností a odeslán do editoru k zobrazení.

5 Vizualizační možnosti

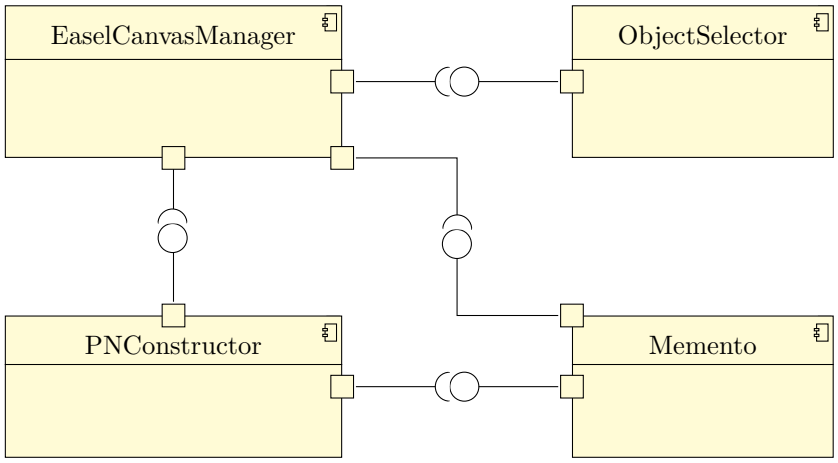
Vizuální část programu je jeho důležitou a nedílnou součástí. Umožňuje přístup k funkcím programu, jednoduché modelování Petriho sítí, názornou simulaci jejich chování či zobrazení výsledků analýzy. Jednou z hlavních součástí této práce je rozšíření vizualizačních možností existujícího programu. V této kapitole se tedy budeme zabývat rozšířeními vizualizační části aplikace.

5.1 Uchovávání stavů

Jedním z vylepšení vizualizačních možností editoru je zapracování možnosti uchovávat jeho stavy a také možnosti libovolně mezi těmito stavy přecházet oběma směry. Uživatel tak má možnost při tvorbě modelu Petriho sítě jednoduše vracet své kroky, což je možnost v dnešní době zcela běžně dostupná například v textových editorech. Tato funkce je v této aplikaci velmi užitečná například při použití automatického layoutu nabízeného editorem. V původním programu došlo po použití automatického layoutu k překreslení vymodelované Petriho sítě podle daného algoritmu, přičemž uživateli nebyl před samotným překreslením nabídnut žádný náhled, aby se mohl rozmyslet, zda chce automatický layout aplikovat. Vzhledem k absenci možnosti návratu do předchozího stavu tak uživatel neměl možnost vrátit se nějakým jednoduchým způsobem k ručně vytvořenému modelu, který se mu mohl zdát z různých důvodů lepší či přehlednější, a zároveň neměl možnost vidět předem náhled, jak bude výsledný layout vypadat.

5.1.1 Implementace

Funkce uchovávání stavů je vyřešena pomocí nového zásuvného modulu *EaselCanvasManagera* *Memento.js* s využitím návrhového vzoru *Memento* popsaného v [18]. Podrobnější popis vzoru *Memento* je v kapitole 5.1.2. Samotný editor má k tomuto modulu přístup přes *PNManagera*, což je potomek *EaselCanvasManagera*. Jeho prostřednictvím získává předcházející, respektive následující stavy, a také informaci, zda lze provést příslušné akce *zpět* a *vpřed*. Dále je pak modul *Memento.js* využíván modulem *PNConstructor*, který při volání některých svých funkcí vytváří nová mementa reprezentující jednotlivé stavy modelované Petriho sítě. *PNConstructor* je modul umožňující tvorbu jednotlivých elementů Petriho sítě při modelování, tedy zajišťuje funkcionalitu například pro přidávání a odebírání míst, přechodů a podobně. Při volání některých těchto funkcí dojde k vytvoření a uložení nového mementa pro možnost pozdějšího využití. Pokud tedy uživatel vytvoří například nové místo v Petriho síti, uloží se tento nový stav do paměti k ostatním mementům. Na obrázku 12 je schematicky znázorněno začlenění modulu *Memento.js* do existujícího programu spolu s několika existujícími zásuvnými moduly.



Obrázek 12: Ukázka začlenění modulu Memento do existujícího programu.

5.1.2 Návrhový vzor memento

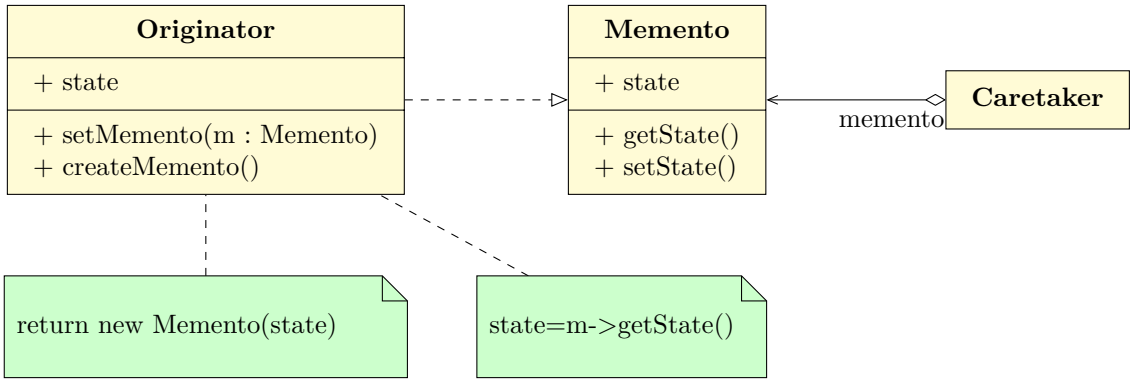
Memento je návrhový vzor, jehož účelem je zachytit a uchovat vnitřní stav objektu bez porušení zapouzdření. Tento stav pak může být použit k obnovení stavu tohoto objektu, kterému se říká memento. Typické využití *Mementa* je například při potřebě vytváření „checkpointů“ či umožnění návratu z chyb nebo testovacích akcí. Návrat objektu do jeho předchozího stavu může být někdy problém, protože tento objekt může mít některé informace uschované jako privátní, a tedy nepřístupné pro ostatní objekty, čímž je znemožněno externí ukládání stavu. Zpřístupnění těchto informací by však narušilo princip zapouzdření. Tento problém řeší právě *Memento*. Jak je vidět na obrázku 13, má tento návrhový vzor tři účastníky - *Memento*, *Caretaker* a *Originator*. Objekt mementa v sobě uchovává „snapshot“ vnitřního stavu tzv. *Originatora*. *Originator* vytvoří snapshot svého stavu včetně privátních informací, který je následně uložen pro pozdější použití. *Originator* je jediný, kdo dokáže s mementy pracovat a získávat z nich informace. Správu jednotlivých mement zajišťuje tzv. *Caretaker*, který v sobě mementa ukládá a poskytuje je na vyžádání zpět *Originatorovi*. *Caretaker* ovšem nikdy nezasahuje do samotných mement a nijak je nemění, jeho úkolem je pouze jejich uchovávání a předávání jiným objektům.

Podle [18] má použití vzoru memento několik důsledků:

- *Zachování zapouzdření* - *Memento* zabraňuje odhalení informací o vnitřním stavu *Originatora*, ke kterým by měl mít přístup pouze *Originator*, ale které je potřeba uložit a uchovávat mimo samotného *Originatora*.

- *Zjednodušení Originatora* - *Originator* nemusí udržovat ve své paměti různé verze svých stavů, čímž se stává jednodušším. Veškerá logika uchovávání a správy mement je mimo *Originatora*.

- *Použití mement může být nákladné* - Vzhledem k tomu, že se mementa uchovávají v paměti, může se stát, že toto uchovávání bude paměťově náročné, pokud *Originator* obsahuje velké



Obrázek 13: Návrhový vzor *Memento* (převzato z [18]).

množství informací, které je potřeba držet v paměti pro pozdější obnovení stavu. Podobně může být náročná samotná operace vytváření mementa, pokud musí *Originator* kopírovat velké množství dat.

- *Skryté náklady na správu mement* - Za správu mement je zodpovědný *Caretaker*, který má tedy i zodpovědnost za jejich mazání. *Caretaker* však neví, jak velká mementa ukládá, takže se může stát, že velmi jednoduchý a nenáročný *Caretaker* bude mít velké paměťové nároky kvůli velikosti uchovávaných mement.

5.2 Nový layout

V původním programu byla možnost využít automatické vykreslení modelované Petriho sítě. Tento layout využívá javascriptovou knihovnu *Dagre* a pomocí jejího layoutovacího algoritmu vytvoří nový layout Petriho sítě podle daného nastavení. Výsledný graf však nemusí vždy vypadat dobře a přehledně, proto jsme se rozhodli přidat do programu další automatický layout. Ten vychází z aplikace *slepého algoritmu* na vykreslování Petriho sítí, která vznikla v rámci semestrálních projektů na VŠB-TUO. Slepý algoritmus je jednoduchý evoluční algoritmus, který náhodně generuje řadu jedinců, ze kterých je nakonec vybrán ten nejlepší. Jako takový se dá aplikovat na generování layoutů Petriho sítě, ze kterých se zvolí ten „nejhezčí“.

Vzhledem k názvu algoritmu, který je základem nového layoutu, používáme pro tento nový layout označení *Blind layout*.

5.2.1 Slepý algoritmus

Evoluční algoritmy jsou popsány v [6] a slepý algoritmus spadá do této třídy algoritmů. Funguje na základě náhodného generování jedinců, což jsou v tomto případě grafy Petriho sítě, z nichž je poté vybrán nejvhodnější jedinec na základě *fitness funkce*. Vygenerování jedince, tedy grafu Petriho sítě, vypadá následovně:

- Jsou určeny hranice, uvnitř kterých se musí vyskytovat všechny vrcholy grafu,

- pro každý přechod i místo sítě jsou pseudonáhodně vygenerovány nové souřadnice, které jsou poté přizpůsobeny mřížce,
- vytvoří se nová reprezentace modelované Petriho sítě, ve které jsou všechna místa i přechody s novými souřadnicemi,
- pro takto vzniklého jedince se vypočítá hodnota jeho *fitness funkce*.

Tímto způsobem je vygenerováno větší množství jedinců, ze kterých je poté vybrán jedinec s nejnižší hodnotou *fitness funkce*. Generování souřadnic do mřížky vede k vizuálně lépe vypadajícím grafům a zvyšuje šanci na vizuálně přijatelnější úhly.

Fitness funkce *Fitness funkce* slouží jako kritérium pro nalezení nejvhodnějšího jedince z vygenerované množiny jedinců. V aplikaci slepého algoritmu na graf Petriho sítě to znamená vizuální ohodnocení grafu. Je proto nutné nějak definovat znaky dobře vypadajícího grafu. V původní aplikaci bylo zohledňováno několik atributů - křížení hran, velikost úhlů a délka hran. V této práci jsme množinu atributů rozšířili o *dostatečnou vzdálenost vrcholů* a *křížení hran s vrcholy*. Pro výpočet hodnoty *fitness funkce* tak zohledňujeme všechny výše zmíněné atributy následujícím způsobem:

- Každé křížení hran způsobí zvýšení hodnoty *fitness*.
- Každé dva vrcholy, které jsou příliš blízko u sebe, způsobí zvýšení *fitness*.
- Každé křížení hran s vrcholy grafu způsobí zvýšení *fitness*.
- Pro délku hran je určen interval optimální délky, ve kterém není hrana ani příliš krátká, ani příliš dlouhá. Za vhodnou délku hrany je hodnota *fitness* snížena, za jinou délku je naopak zvýšena.
- V hodnocení velikosti úhlů je zvyšována hodnota *fitness* pro velmi ostré úhly. Pro vizuálně pěkné úhly (90°, 45° nebo 180°) je naopak hodnota *fitness* snížena.

Aplikací výše uvedených pravidel dochází k preferování grafů, ve kterých jsou vrcholy dostatečně vzdáleny od sebe, hrany se nekříží se žádnými vrcholy ani s jinými hranami, mají přiměřenou délku a obsahují vizuálně pěkné úhly.

5.2.2 Implementace

Při implementaci *Blind layoutu* jsme se rozhodli zvolit odlišný přístup, než jaký je použit pro existující *Dagre layout*. Místo generování nových souřadnic přímo v prohlížeči pomocí *WebWorkerů* provádíme výpočty na analytickém serveru. Hlavním důvodem pro toto řešení byl fakt, že generování velkého množství grafů při běhu slepého algoritmu může být výpočetně náročné, a proto se jeví jako optimální provést tyto výpočty na serveru a ne přímo v prohlížeči.

Editor Hlavní změnou v editoru bylo přidání nového souboru *PNBlindLayout.js*, ve kterém probíhá vyžádání nového layoutu od analytického serveru a jeho následná aplikace na aktuální modelovanou Petriho síť. Implementovaná aplikace slepého algoritmu mění pouze souřadnice míst a přechodů sítě, nijak ovšem nepracuje s hranami. V důsledku této skutečnosti proto dochází k nastavení nových souřadnic příslušným místům a přechodům modelované sítě. Jednotlivým hranám je nastaven pouze výchozí a koncový bod. Tyto body odpovídají zdrojovému a cílovému objektu, které hrana spojuje. Jakékoliv dříve existující pomocné body jsou tímto z hran odstraněny a ze všech hran grafu se tak stanou úsečky. Uživatel si pak samozřejmě může graf dále manuálně upravit, včetně přidání pomocných bodů na hranách.

Editor zde konzumuje RESTful webovou službu analytického serveru, odesílá na server informace o modelované Petriho síti v JSON podobě a očekává opět Petriho síť v JSON formátu, obsahující informace o nových souřadnicích jednotlivých míst a přechodů.

Implementace dále vyžadovala ještě několik menších úprav existujícího kódu, aby byla zajištěna potřebná funkcionalita.

Analytický server Většina funkcionality *Blind layoutu* probíhá na analytickém serveru. Za účelem zajištění komunikace s editorem bylo nutné vytvořit novou implementaci serializace Petriho sítě do *JSON* formátu. Tento serializátor se nachází ve třídě *PNModule*, která v původní verzi umožňovala deserializaci Petriho sítě z JSON formátu do objektové reprezentace v jazyce *Java*. Serializují se pouze údaje nutné pro vykreslení Petriho sítě v editoru, tedy ID a souřadnice jednotlivých objektů.

Třída *BlindLayoutGenerator* se nachází v balíku *cz.diploma.server.api* a funguje jako prostředník pro komunikaci mezi serverem a editorem pro generování layoutu. Vystupuje jako další zdroj analytického serveru, který mohou konzumovat jeho klienti, a zpřístupňuje tak klientům automatický layout. Obsahuje jedinou metodu *blindLayout()* a příslušné anotace, díky kterým je komunikace zprostředkována. Jako vstupní parametr očekává Petriho síť a vrací nově vygenerovanou síť vytvořenou slepým algoritmem. Tato Petriho síť ovšem není úplnou objektovou reprezentací modelované Petriho sítě, protože obsahuje pouze informace nutné k jejímu vykreslení.

Samotná implementace slepého algoritmu se nachází v balíku *cz.diploma.server.layout*. Jedná se o tři třídy zajišťující veškerou potřebnou funkcionalitu. Třída *BlindAlgorithm* obsahuje metody pro výpočet a nastavení hranic generovaných grafů, generování samotných jedinců a nalezení jedince s nejnižší hodnotou *fitness funkce*. Oproti původní aplikaci slepého algoritmu je zde navíc přidána funkcionalita, zajišťující minimální rozměry ohraničení grafu v závislosti na počtu objektů. Nemůže se tak stát, že by se potenciální rozměry grafu opakovaným použitím *Blind layoutu* neustále zmenšovaly. Další modifikací je závislost velikosti mřížky, do které jsou umisťovány objekty Petriho sítě, na počtu objektů sítě. Při vygenerování jedince je vždy vypočtena i příslušná hodnota *fitness funkce*, k čemuž slouží metody třídy *FitnessCalculator*. Tato třída

obsahuje veškeré potřebné metody pro výpočet hodnoty *fitness* daného grafu. Třetí třída *IndividualGraph* slouží jako reprezentace jedince slepého algoritmu.

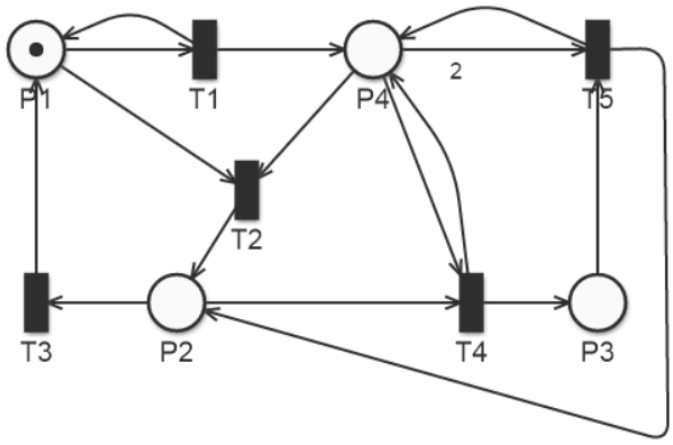
Skutečnost, že je mezi vygenerované jedince vždy přidán i původní graf, zaručuje, že pokud algoritmus nevygeneruje vizuálně lepší graf, bude výsledným vybraným jedincem právě původní graf. Nedojde tedy ke zhoršení vizuální kvality grafu, alespoň vzhledem k nastaveným ohodnocením pro *fitness funkci*. Důsledkem této skutečnosti je možnost opakovaného použití *Blind layoutu* za účelem nalezení nejlepšího vygenerovatelného grafu, protože při opětovném použití bude hodnota *fitness* nového grafu vždy stejná nebo lepší než hodnota grafu původního.

6 Ukázky a porovnání

V této kapitole se budeme zabývat ukázkami nově přidaných možností programu pro editaci a analýzu Petriho sítí a také porovnáním nových analytických metod s metodami existujícími v původní verzi programu.

6.1 Přínos NMRT

V původním programu je jednou z analytických metod používaných pro testování vlastností Petriho sítí generování stromu dosažitelnosti, respektive pokrytí. Pokud je analyzovaná Petriho síť neomezená, je vygenerován *líný strom pokrytí*, který částečně uchovává informace o stavu tokenů souběžně s informací o neomezenosti daného místa. Tento přístup je použitelný na všechny třídy Petriho sítí, protože vlastně částečně generuje strom dosažitelnosti. Nevýhodou je ovšem fakt, že při testování vlastností na základě *líného stromu pokrytí* může dojít k chybám. Příkladem takovéto situace je síť z obrázku 14. Na obrázku 15 jsou výsledky analýzy této sítě bez použití NMRT. Program vyhodnotil síť jako živou, reverzibilní a bez uzamčení. Velmi jednoduše lze ovšem dokázat, že v této Petriho síti existuje uzamčení, takže nemůže být ani živá ani reverzibilní. Provedením jednoduché sekvence přechodů $\sigma = T_1 \rightarrow T_1 \rightarrow T_2 \rightarrow T_4$ se síť bude nacházet ve značení (0,0,1,1), ve kterém není žádný přechod proveditelný, a jedná se tedy o uzamčení.

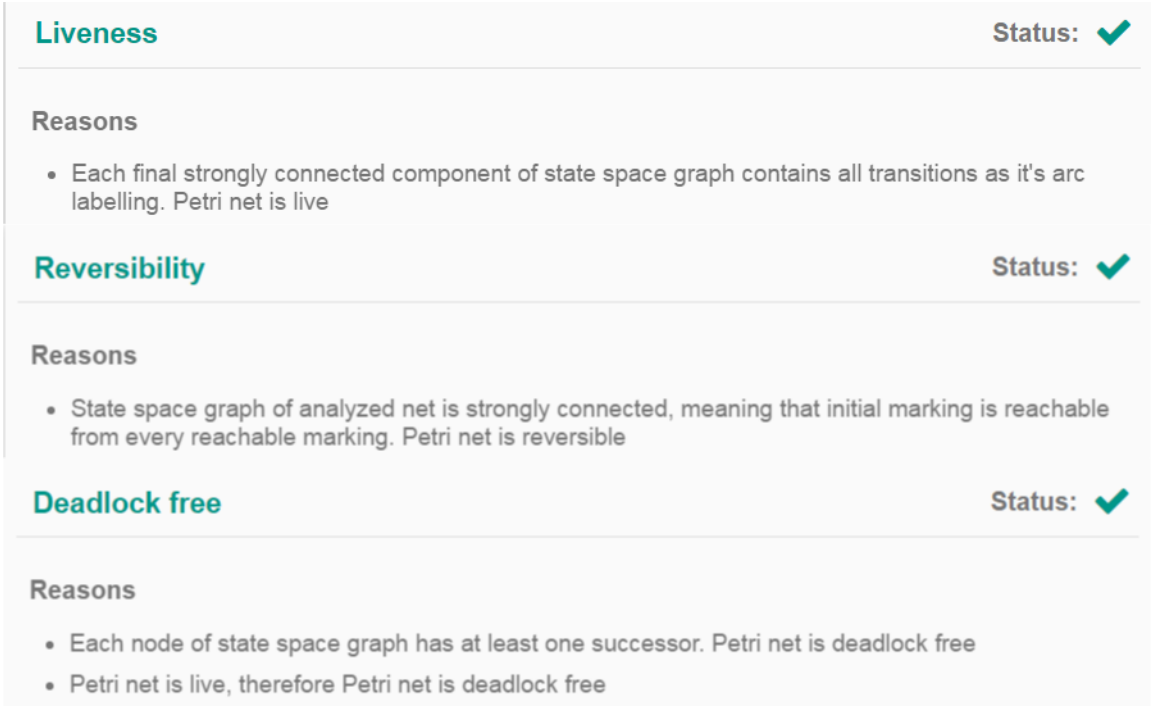


Obrázek 14: Petriho síť obsahující uzamčení (screenshot z rozšiřovaného programu).

Na obrázku 16 je výsledek analýzy Petriho sítě z obrázku 14 s použitím NMRT. Jak je na obrázku vidět, program pomocí NMRT nalezl uzamčení a díky tomu správně vyhodnotil vlastnosti Petriho sítě.

6.1.1 Ukázka testu dosažitelnosti

Test dosažitelnosti umožňuje uživateli zjistit, zda jím zvolené značení je v modelované Petriho síti dosažitelné či nikoliv. Ukázku předvedeme na jednoduché Petriho síti z obrázku 17. Jak je



Obrázek 15: Výsledky analýzy Petriho sítě z obrázku 14 bez použití NMRT.

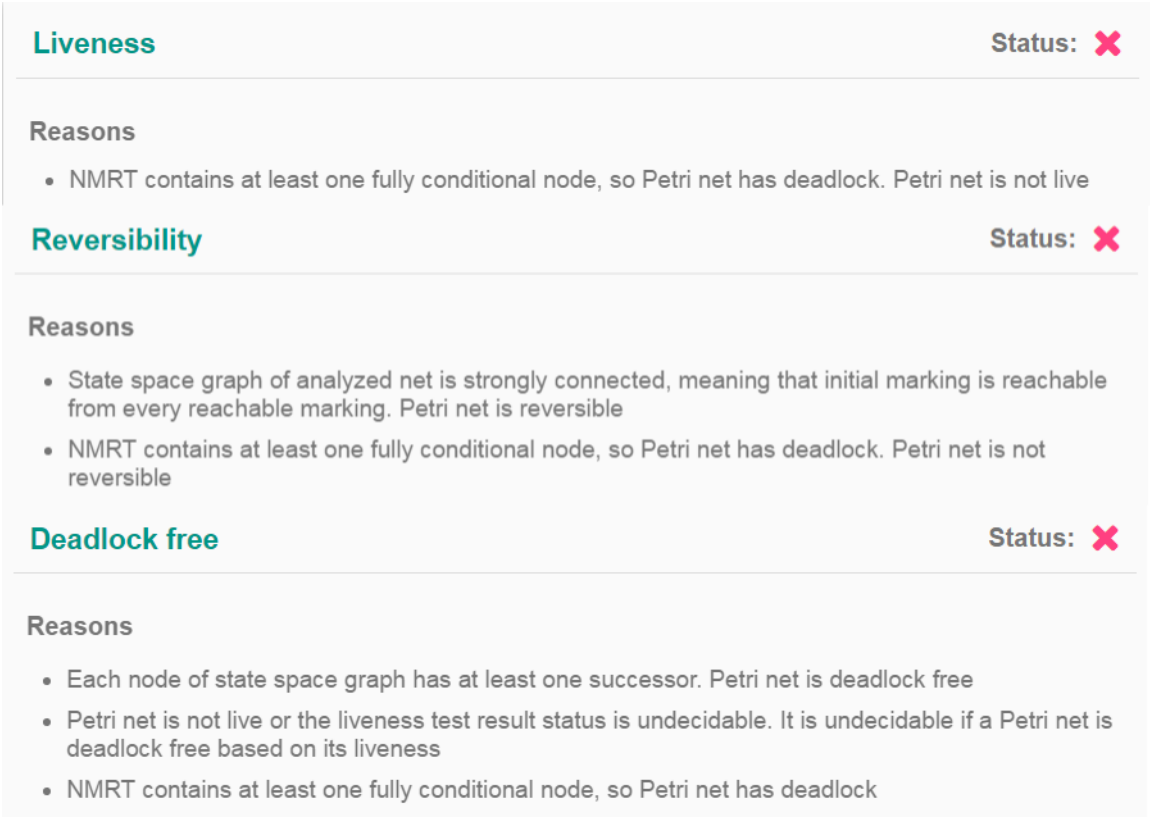
na obrázku vidět, síť obsahuje dvě neomezená místa *P2* a *P3*, přičemž v místě *P2* přibývají tokeny vždy po dvou.

Na obrázcích 18 a 19 jsou vidět ukázkové výsledky testu. V prvním případě je značení zjevně nedosažitelné, protože v místě *P2* může být lichý počet tokenů pouze v případě, že v místě *P1* již po provedení přechodu *T3* není žádný token. Na druhou stranu značení z druhého obrázku je dosažitelné, přestože požadovaný počet tokenů v místě *P2* je relativně vysoký.

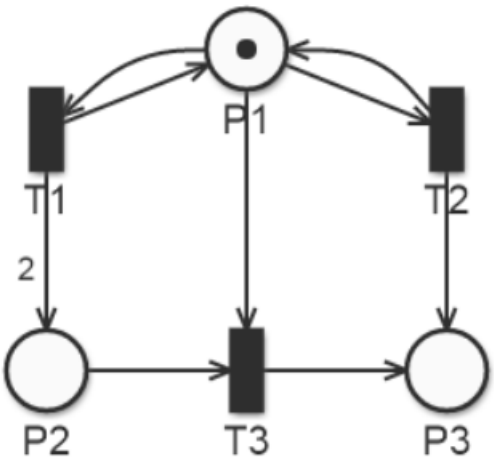
6.2 Ukázka použití Blind layoutu

Blind layout slouží jako další alternativa k již existujícímu *Dagre layoutu*. Oba tyto automatické layouty slouží k automatickému vykreslení modelované Petriho sítě, přičemž každý z nich používá jiný postup. Zatímco *Dagre layout* vypočítává nové souřadnice objektů i hran přesně definovanou heuristikou, *Blind layout* využívá pseudonáhodné generování jedinců, z nichž vybere nejlepšího na základě *fitness* funkce. Vzhledem k absenci heuristiky je *Blind layout* vhodný především pro použití na jednodušší modely Petriho sítí, kde jeho aplikace vede k vizuálně pěkným grafům. *Dagre layout* je díky své propracované heuristice vhodný zejména pro složitější modely Petriho sítí, ve kterých je vysoký poměr hran a objektů.

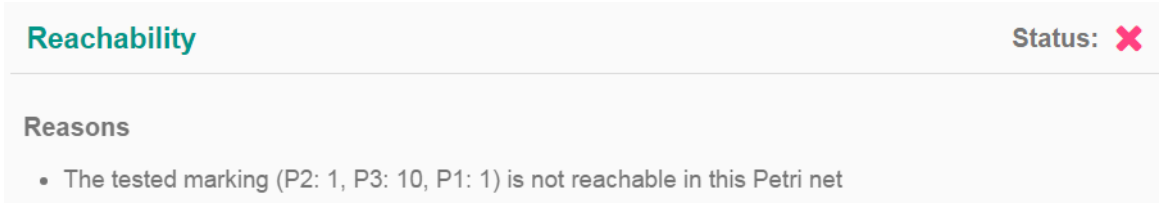
Na obrázku 20 je vidět ručně vytvořená Petriho síť. Výsledek po použití *Blind layoutu* je na obrázku 21. Výsledný layout lze určitě ohodnotit jako dobře vypadající vzhledem k používaným kritériím. Při aplikaci *Blind layoutu* na složitější grafy obvykle dochází k vizuálně horším výsledkům.



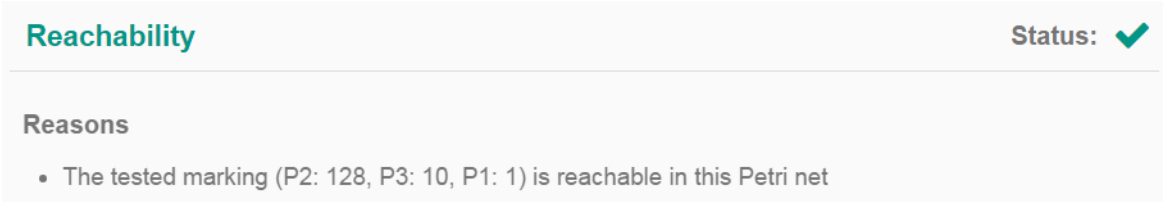
Obrázek 16: Výsledky analýzy Petriho sítě z obrázku 14 s použitím NMRT.



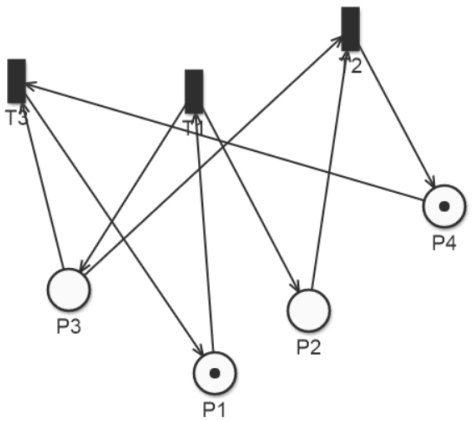
Obrázek 17: Ukázková Petriho síť pro test dosažitelnosti.



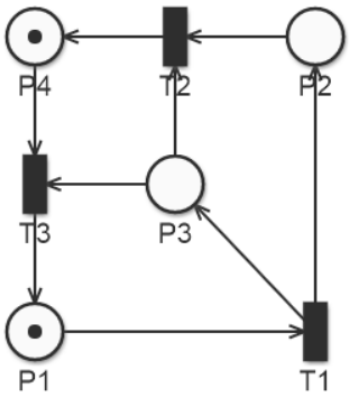
Obrázek 18: Ukázková Petriho síť pro test dosažitelnosti.



Obrázek 19: Ukázková Petriho síť pro test dosažitelnosti.



Obrázek 20: Vzhled Petriho sítě před použitím *Blind layoutu* (screenshot).



Obrázek 21: Vzhled Petriho sítě po použití *Blind layoutu* (screenshot).

7 Závěr

Cílem této diplomové práce bylo především rozšířit program pro editaci a analýzu P/T Petriho sítí o nové vizualizační a analytické možnosti. Vzhledem ke komplexnosti a rozsáhlosti původního programu bylo nutné se především důkladně seznámit s jeho možnostmi, způsoby řešení konkrétních problémů a možnostmi implementace nových komponent. Tuto nutnou součást rozšiřování existujícího programu bohužel značně komplikovala absence programátorské dokumentace či alespoň komentářů v kódu.

Dalším cílem pak bylo vytvořit přehled grafových metod použitelných pro analýzu Petriho sítí. Mezi námi uvedené metody patří především běžně používané způsoby analýzy. Kromě nich zmiňujeme také několik dalších metod, které lze použít. U některých uvedených metod se však jejich použití nejeví jako efektivní, například kvůli velké časové a paměťové náročnosti. V takovém případě uvádíme také alternativní způsob analýzy, který lze použít. Příkladem takové situace je například určení konzervativnosti, protože test s využitím stromu dosažitelnosti vede k problému lineárního programování, jehož náročnost roste s počtem vrcholů stromu dosažitelnosti.

Po vizuální stránce považujeme původní program za dobře zpracovaný. Do programu jsme proto přidali jen několik vylepšení pro uživatele. Prvním z nich je možnost přecházení mezi různými stavy modelu. Funkce návratu zpět či vpřed je běžnou součástí nejrůznějších editorů, ať už textových nebo grafických. Tuto funkcionalitu jsme implementovali formou nového zásuvného modulu *EaselCanvasManagera*. Jedná se o naši implementaci návrhového vzoru *Memento*. Druhým významným vylepšením je přidání nového automatického layoutu, který jsme nazvali *Blind layout*. Základem tohoto layoutu je *slepý evoluční algoritmus*, jehož aplikaci na model Petriho sítě jsme použili v této práci. Uživatel tak má k dispozici další možnost automatického vykreslení modelované Petriho sítě, která podle nás dosahuje dobrých výsledků především pro jednoduché modely.

Pro rozšíření analytických možností programu jsme zvolili implementaci New Modified Reachability Tree. Jedná se o konečný strom dosažitelnosti pro neomezené sítě. Přestože jej nelze aplikovat bez rizika chybných výsledků na celou množinu Petriho sítí, ale jen na její část, možnosti jeho aplikace jsou zajisté větší než možnosti běžně používaných metod. Naši implementaci tohoto stromu dosažitelnosti pak využíváme při ověřování vlastností Petriho sítí jako jsou živost či reverzibilita. Také nám to umožnilo testování dosažitelnosti značení, a to i pro neomezené Petriho sítě. Testování dosažitelnosti značení nebylo v původní práci vůbec implementováno.

Při implementování rozšíření programu jsme se snažili dodržovat zásady a principy, které byly zaváděny v původním programu. Snažili jsme se zachovat maximální rozšiřitelnost všech komponent programu. Naším cílem také bylo udržení použitelnosti všech možností programu pro běžného uživatele, především s ohledem na časovou náročnost analytických a vizualizačních metod. Vzhledem k potenciálnímu dalšímu rozšiřování programu jsme se také snažili o usnadnění pochopení nově přidaného kódu. Domníváme se, že tyto cíle se nám podařilo naplnit.

V rámci dalšího vývoje programu by jistě mohlo dojít k rozšíření množiny testovaných vlastností a množiny analytických metod. Program umožňuje jejich snadné přidání a začlenění do systému. Možným rozšířením může být také přidání dalších algoritmů pro automatické vykreslování modelu Petriho sítě, zejména pro složitější modely, pro které námi implementovaný *Blind layout* není příliš vhodný. Vzhledem ke stále probíhajícímu výzkumu v oblasti analýzy neomezených Petriho sítí je zajisté variantou dalšího vylepšení programu aplikace nových metod pro analýzu neomezených Petriho sítí. Může se jednat i o rozšíření využitelnosti New Modified Reachability Tree.

Literatura

[1] DESROCHERS, Alan A.; AL-JAAR, Robert Y. *Applications of Petri nets in manufacturing systems: modeling, control, and performance analysis*. IEEE, 1995.

[2] EVEN, Shimon. *Graph algorithms*. Cambridge University Press, 2011.

[3] DER JENG, Mu; PENG, Mao Yu. *On the liveness problem of 1-place-unbounded Petri nets*. In: Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on. IEEE, 1997. p. 3221-3226.

[4] DER JENG, Mu; PENG, Mao Yu. *Augmented reachability trees for 1-place-unbounded generalized Petri nets*. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 1999, 29.2: 173-183.

[5] JOHNSON, Donald B. *Finding all the elementary circuits of a directed graph*. SIAM Journal on Computing, 1975, 4.1: 77-84.

[6] JONES, Terry. *Evolutionary algorithms, fitness landscapes and search*. 1995. PhD Thesis. The University of New Mexico.

[7] KARP, Richard M.; MILLER, Raymond E. *Parallel program schemata*. Journal of Computer and system Sciences, 1969, 3.2: 147-195.

[8] KOVÁŘ, Petr. *Teorie Grafů* [elektronická skripta]. Text dostupný z: *http://homel.vsb.cz/~kov16/files/skriptum_teorie_grafu_tisk.pdf*

[9] LEE, Chin Yang. *An algorithm for path connections and its applications*. IRE transactions on electronic computers, 1961, 3: 346-365.

[10] LEISERSON, Charles E.; SCHARDL, Tao B. *A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers)*. In: Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures. ACM, 2010. p. 303-314.

[11] MARKL, Jaroslav. *Petriho sítě I* [elektronická skripta]. Text dostupný z: *http://drazdilova.cs.vsb.cz/Data/Sites/5/petrinet/petrinetsylabus.pdf*

[12] MOORE, Edward F. *The shortest path through a maze*. Bell Telephone System., 1959.

[13] MURATA, Tadao. *Petri nets: Properties, analysis and applications*. Proceedings of the IEEE, 1989, 77.4: 541-580.

[14] NUUTILA, Esko; SOISALON-SOININEN, Eljas. *On finding the strongly connected components in a directed graph*. Information Processing Letters, 1994, 49.1: 9-14.

[15] RU, Yu; WU, Weimin; HADJICOSTIS, Christoforos N. *"Comments on"A Modified Reachability Tree Approach to Analysis of Unbounded Petri Nets*. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 2006, 36.5: 1210-1210.

[16] SKIENA, Steven S. *The algorithm design manual*. Springer Science & Business Media, 1998.

[17] TARJAN, Robert. *Depth-first search and linear graph algorithms*. SIAM journal on computing, 1972, 1.2: 146-160.

[18] VLISSIDES, John, et al. *Design patterns: Elements of reusable object-oriented software*. Reading: Addison-Wesley, 1995, 49.120: 11.

[19] WANG, F.-Y. *A modified reachability tree for Petri nets*. In: Systems, Man, and Cybernetics, 1991.'Decision Aiding for Complex Systems, Conference Proceedings., 1991 IEEE International Conference on. IEEE, 1991. p. 329-334.

[20] WANG, Fei-Yue; GAO, Yanqing; ZHOU, MengChu. *A modified reachability tree approach to analysis of unbounded Petri nets*. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 2004, 34.1: 303-308.

[21] WANG, ShouGuang, et al. *A new modified reachability tree approach and its applications to unbounded Petri nets*. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2013, 43.4: 932-940.

[22] WANG, Shouguang, et al. *A reduced reachability tree for a class of unbounded Petri nets*. IEEE/CAA Journal of Automatica Sinica, 2015, 2.4: 345-352.

[23] YANG, Ru, et al. *Liveness Analysis of ω-Independent Petri Nets Based on New Modified Reachability Trees*. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2016.

[24] YILMAZ, Buket. *Applications of Petri nets*. 2008. PhD Thesis. Izmir Institute of Technology.

A Uživatelský manuál

K práci přikládáme uživatelský manuál z původní práce. Obsahuje stručné návody k použití jednotlivých komponent nástroje včetně ukázek. Manuál se nachází na přiloženém CD.

B Poznámky k instalaci a údržbě

Přestože k původní práci existuje uživatelský manuál obsahující mimo jiné také návod na instalaci všech hlavních komponent nástroje, některé informace v něm chybí. Pro usnadnění instalace či udržování v provozu na serveru jsme se rozhodli připojit k této práci i několik poznámek a dodatků k existujícímu manuálu.

Pro sestavení editoru lze postupovat podle manuálu. Zde pouze upozorníme, že je nutné mít na daném zařízení nainstalovaný také Git, pomocí kterého nástroje pro automatizaci stahují důležité soubory. Jinak však lze pomocí manuálu sestavit editor oběma uvedenými způsoby, tedy ve verzi pro vývoj i ve verzi pro nasazení do produkčního prostředí.

Analytický server a databázové úložiště je možné sestavit pomocí nástroje Maven. Za tímto účelem doporučujeme používat IDE Netbeans. Pro správné fungování aplikace je důležité pojmenování výsledných souborů webového archivu. Editor je konfigurován tak, že očekává konkrétní pojmenování u zbývajících komponent nástroje. V době psaní této práce to byly názvy *analysis.war* a *projectStorage.war*. V případě změny těchto názvů je nezbytné provést i odpovídající úpravy v konfiguračním souboru editoru, jinak nebude vzájemná komunikace fungovat.

Pro samotné nasazení doporučujeme použít *Apache Tomcat* verze 7. K dispozici je také *Apache Tomcat service*, což při nasazení na server poskytuje výhodu v možnosti nastavení automatického spouštění. V době psaní této práce byl nástroj dostupný na adrese *edukin.vsb.cz:8082/petrineteditor/*. Apache Tomcat fungoval na portu 8082, v případě změny je nutné příslušně upravit také adresy v konfiguračním souboru editoru.

Pro běh databázového úložiště je nutné, aby na cílovém zařízení byla nainstalována příslušná databáze. Za tímto účelem doporučujeme MySQL. V době psaní této práce byla na serveru databáze s údaji popsanými již v uživatelském manuálu k původní práci.